

APPENDIX A : An EMBODIMENT OF RDL GRAMMAR

This section describes an embodiment of the RDL in a format that can be directly passed to a *yacc* tool. The bold letters are grammar keywords and the rest are the grammar rules. The grammar describes the allowed syntax for describing the target architecture.

RDL Grammar:

Top : Top Defs
 | Defs

Defs : UnitDef
 | ResourceDef
 | FunctionalityDef
 | ParameterDef

ParameterDef : **PARAMETER** Name INTEGER

Unitdef : **UNITDEF** Name
 {
 UnitBody
 }

UnitBody : UnitInstantiationsList StatementList

UnitInstantiationsList : UnitInstantiationsList UnitInstantiations
 | UnitInstantiations

UnitInstantiations : UnitInstance
 | ResourceInstance

ResourceInstance : **RESOURCE** Name ArrayNameList

ArrayNameList : ArrayNameList , ArrayName
| ArrayName

ArrayName : ArrayName | ArithExpr |
| ArrayName (ArithExpr)
| Name

UnitInstance : **UNIT** Name ArrayNameList

BasicStmt : RconnectStmt
| UseStmt

RconnectStmt : **RCONNECT**(ArrayNameList) ;

UseStmt : **USES**(ArrayNameList) ;

FunctionalityDef : **FUNCTIONALITYDEF** Name
{
 FuncDeclarationList
 FuncConnectionList
}

FunctionDeclarationList : FunctionDeclarationList FuncDeclaration
| FuncDeclaration

FunctionConnectionList : FunctionConnectionList FuncConnection
| FuncConnection

NameList : NameList , Name
| Name

FuncDeclaration : **INPUT** .NameList ;
| **INPUT_OPT** NameList ;
| **OUTPUT** NameList ;
| **OUTPUT_OPT** NameList ;
| Name NameList ;

FuncConnection : **DCONNECT**(Name , Name) ;

ResourceDef : **RESOURCEDEF** Name

```
{  
    Functionality_Supported  
    Wrapper  
}
```

Functionality_Supported : **FUNCTIONALITY** NameList ;

Wrapper : Wrapper Attribute
| Attribute

Attribute : **ATTRIBUTE** Name

```
{  
    AttributeBody  
}
```

AttributeBody : Cminus
| FuncBasedAttributeList

FuncBasedAttributeList : FuncBasedAttributeList FuncBasedAttribute
| FuncBasedAttribute

FuncBasedAttribute : < Name > Cminus

Cminus : { CminusDeclrList StatementList }
| { CminusDeclrList }
| { StatementList }

CminusDeclrList : CminusDeclrList CminusDeclr
| CminusDeclr

CminusDeclr : **float** NameList
| **int** NameList
| **table** NameList

Statement : IfStmt

LogicalExpr : LogicalExpr == LogicalExpr
 | LogicalExpr != LogicalExpr
 | LogicalExpr < LogicalExpr
 | LogicalExpr <= LogicalExpr
 | LogicalExpr >= LogicalExpr
 | LogicalExpr > LogicalExpr
 | LogicalExpr || LogicalExpr
 | LogicalExpr && LogicalExpr
 | LogicalExpr
 | ArithExpr

DataType : INTEGER
 | FLOATINGPOINT
 | STRING
 | CompositeName

CompositeName : CompositeName — > ArrayName
 | ArrayName

Name : IDENTIFIER

APPENDIX B **An RDL specification for Xilinx Virtex2 XC2V250**

```

5  /*******
   ***
   /**      COPYRIGHT (C), 2001
   /**
   /**      Company ..... AccelChip Inc.
   /**      999 Plaza Drive, Suite 340, Schaumburg, IL 60173
10  /**
   /**      Program ..... Description of the Xilinx
   /**                                  Virtex2 XC2V250
   /**                                  architecture in ACCEL RDL
   /**
15  /**
   /*******
   ***/

   //////////////////////////////////////
20  /
   /** This is the RDL file for Xilinx's Virtex2 device XC2V250
   /**
   /** This device has 24 SelectRAM memory blocks with storage for 432 Kb
   /**
25  /** of data. The location properties use the following form :
   /**
   /** LOC = RAMB16_X#Y#. It has 24 multipliers arranged in 4 columns with
   /**
   /** 6 multipliers per column. The location properties use the
30  /**
   /** following form : LOC = MULT18X18_X#Y#
   /**
   /** The Logic Cells are arranged in a 24 x 16 grid
   /**
35  //////////////////////////////////////
   /

   PARAMETER NUM_MULT_COLUMNS 4;
   PARAMETER NUM_EMBED_MULT_PER_COLUMN 6;
40  PARAMETER NUM_RAM 24;
   PARAMETER NUM_LOGIC_ROWS 24;
   PARAMETER NUM_LOGIC_COLUMNS 16;

   RESOURCEDEF GLOBAL
45  {
   //////////////////////////////////////
   /**

   FUNCTIONALITY GLOBALDECLARATIONS;

50  ATTRIBUTE INSTANTIATION
   {

```

```

5  //////////////////////////////////////////////////
   // Component Declaration for the Xilinx //
   // embedded multipliers                //
   //////////////////////////////////////////////////

   attribute +      "void MULT18X18(input wire[18] A,";
   attribute +      "input wire[18] B,";
   attribute +      "output wire[36] P";
10  attribute +      "); \n";

   //////////////////////////////////////////////////
   // Component instantiation for the Xilinx//
   // Block SelectRam with Dual Port      //
15  // A:16,384 x 1 bit and B:16,384 x 1 bit //
   //////////////////////////////////////////////////

   attribute +      "void RAMB16_S1_S1(input wire[1] DIA,";
   attribute +      "input wire[14] ADDRA,";
20  attribute +      "input wire[1] ENA,";
   attribute +      "input wire[1] WEA,";
   attribute +      "input wire[1] SSRA,";
   attribute +      "input wire[1] CLKA,";
   attribute +      "output wire[1] DOA,";

25  attribute +      "input wire[1] DIB,";
   attribute +      "input wire[14] ADDRFB,";
   attribute +      "input wire[1] ENB,";
   attribute +      "input wire[1] WEB,";
30  attribute +      "input wire[1] SSRB,";
   attribute +      "input wire[1] CLKB,";
   attribute +      "output wire[1] DOB";

   attribute +      "); \n";

35  //////////////////////////////////////////////////
   // Component instantiation for the Xilinx//
   // Block SelectRam with Dual Port      //
   // A:8192 x 2 bits and B:8192 x 2 bits  //
40  //////////////////////////////////////////////////

   attribute +      "void RAMB16_S2_S2(input wire[2] DIA,";
   attribute +      "input wire[13] ADDRA,";
   attribute +      "input wire[1] ENA,";
45  attribute +      "input wire[1] WEA,";
   attribute +      "input wire[1] SSRA,";
   attribute +      "input wire[1] CLKA,";
   attribute +      "output wire[2] DOA,";

50  attribute +      "input wire[2] DIB,";
   attribute +      "input wire[13] ADDRFB,";
   attribute +      "input wire[1] ENB,";
   attribute +      "input wire[1] WEB,";
   attribute +      "input wire[1] SSRB,";

```

```

attribute +      "input wire[1] CLKB,";
attribute +      "output wire[2] DOB";

attribute +      "); \n";
5
////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port      //
// A:4096 x 4 bits and B:4096 x 4 bits  //
10
////////////////////////////////////

attribute +      "void RAMB16_S4_S4(input wire[4] DIA,";
attribute +      "input wire[12] ADDRA,";
attribute +      "input wire[1] ENA,";
15
attribute +      "input wire[1] WEA,";
attribute +      "input wire[1] SSRA,";
attribute +      "input wire[1] CLKA,";
attribute +      "output wire[4] DOA,";

20
attribute +      "input wire[4] DIB,";
attribute +      "input wire[12] ADDR8B,";
attribute +      "input wire[1] ENB,";
attribute +      "input wire[1] WEB,";
25
attribute +      "input wire[1] SSRB,";
attribute +      "input wire[1] CLKB,";
attribute +      "output wire[4] DOB";

attribute +      "); \n";

30
////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port      //
// A:2048 x 9 bits and B:2048 x 9 bits  //
35
////////////////////////////////////

attribute +      "void RAMB16_S9_S9(input wire[8] DIA,";
attribute +      "input wire[1] DIP8A,";
attribute +      "input wire[11] ADDRA,";
40
attribute +      "input wire[1] ENA,";
attribute +      "input wire[1] WEA,";
attribute +      "input wire[1] SSRA,";
attribute +      "input wire[1] CLKA,";
attribute +      "output wire[8] DOA,";
45
attribute +      "output wire[1] DOPA,";

attribute +      "input wire[8] DIB,";
attribute +      "input wire[1] DIPB,";
attribute +      "input wire[11] ADDR8B,";
50
attribute +      "input wire[1] ENB,";
attribute +      "input wire[1] WEB,";
attribute +      "input wire[1] SSRB,";
attribute +      "input wire[1] CLKB,";
attribute +      "output wire[8] DOB,";

```



```

RESOURCEDEF UNSIGNED_DIVIDER
{
    FUNCTIONALITY DIV;
5    ATTRIBUTE SOFTCORE
    {
        <DIV>
        {
            attribute + "true";
10        }
    }
    ATTRIBUTE INSTANTIATION
    {
        <DIV>
15        {
            int before_dec;
            int after_dec;
            before_dec = 0;
            after_dec = 0;

20            after_dec = in1->fractional_bits();
            before_dec = 16 - after_dec;

            attribute +
25            " input wrap unsigned fixed [" + before_dec + "," +
after_dec + "]" + component_name + "_dividend;\n" +
            " input wrap unsigned fixed [" + before_dec + "," +
after_dec + "]" + component_name + "_divisor;\n" +

30            " output wrap unsigned fixed [16,0] " +
component_name + "_quot;\n" +
            " output wrap unsigned fixed [1,15] " +
component_name + "_remd;\n";
            attribute +
35            " unsigned wrap fixed [7,0] " + component_name +
"_latency;\n";

            attribute +
            " instantiate UNSIGNED_DIVIDER : " + component_name
40 +
            "(" +
            " + component_name + "_dividend," +
            " divisor = " + component_name +
            "_divisor," +
45            " quot = " + component_name + "_quot,"
            +
            " remd = " + component_name + "_remd," +
            " c = " + clock_name +
50            " );";
        }
    }
}

ATTRIBUTE CAN_DO

```

```

{
    <DIV>
    {
        if ( in1->bitwidth( )<17 && in2->bitwidth( )<17 && in2-
5  >bitwidth( )>2 &&
                                in1->is_unsigned( ) == true &&
                                in2->is_unsigned( ) == true )
        {
10             attribute + "true";
        }
        else
        {
15             attribute + "false";
        }
    }
}

ATTRIBUTE PIPE_DELAY
20 {
    <DIV>
    {
        attribute + "1";
    }
25 }

ATTRIBUTE COMBINATIONAL
{
    <DIV>
30 {
        attribute + "false";
    }
}

35 ATTRIBUTE NUM_STATES
{
    <DIV>
    {
40         attribute + "2";
    }
}

ATTRIBUTE INTERFACE
45 {
    <DIV>
    {
        int fract;
        int start_position_quot;
        int end_position_remd;
50     int mid_position_result;
        fract = out1->fractional_bits();
        start_position_quot = 15 - fract;
        end_position_remd = 16 -fract;
        mid_position_result = fract -1;
    }
}

```

```

        attribute + "state1: {" ;
        attribute + component_name + "_dividend = " + in1->name( ) + "
; \n";
        attribute + component_name + "_divisor = " + in2->name( ) + "
5 ; \n";
        attribute + component_name + "_latency = 0 ; \n";
        attribute + " goto state2 ; \n";
        attribute + "};\n";
        attribute + "state2: {" ;
10 attribute + " if (" + component_name + "_latency " + " >20
) \n";
        attribute + " { " + out1->name( ) + "(15:" + fract + " )
= " +
component_name + "_quot( " + start_position_quot +
15 " :0 ) ; \n";
        attribute + " " + out1->name( ) + "( " +
mid_position_result + ":0 ) ="
+ component_name + "_remd ( 15:" +
end_position_remd +
20 " ) ; \n";
        attribute + " goto NEXTSTATE; \n";
        attribute + " } \n";
        attribute + " else \n";
        attribute + " { " + component_name + "_latency = " +
25 component_name + "_latency + 1 ; \n";
        attribute + " } \n";
        attribute + "};\n";
    }
}
30 }

////////////////////////////////////
// Signed Division IP Core //
35 //////////////////////////////////////

RESOURCEDEF SIGNED_DIVIDER
{
    FUNCTIONALITY DIV;
    ATTRIBUTE SOFTCORE
40 {
        <DIV>
        {
            attribute + "true";
45        }
    }
    ATTRIBUTE INSTANTIATION
    {
        <DIV>
        {
50 int before_dec;
        int after_dec;
        before_dec = 0;
        after_dec = 0;
        after_dec = in1->fractional_bits();
    }
}

```

```

        before_dec = 16 - after_dec;
        attribute +
            " input wrap signed fixed [" + before_dec + "," +
after_dec+"] " + component_name + "_dividend;\n" +
5         " input wrap signed fixed [" + before_dec + "," +
after_dec+"] " + component_name + "_divisor;\n" +
            " output wrap signed fixed [16,0] " + component_name
+ "_quot;\n" +
            " output wrap signed fixed [1,15] " + component_name
10 + "_remd;\n";
        attribute +
            " unsigned wrap fixed [7,0] " + component_name +
"_latency;\n";

15         attribute +
            " instantiate SIGNED_DIVIDER : " + component_name +
            "(" +
            " dividend =
" + component_name + "_dividend," +
            " divisor = " + component_name +
20 "_divisor," +
            " quot = " + component_name + "_quot,"
+
            " remd = " + component_name + "_remd," +
            " c = " + clock_name +
25 ");";
    }
}

30 ATTRIBUTE CAN_DO
{
    <DIV>
    {
        if ( in1->bitwidth( )<17 && in2->bitwidth( )<17 && in2-
35 >bitwidth( )>2 &&
            in1->is_signed( ) == true && in2-
>is_signed( ) == true )
        {
            attribute + "true";
        }
        else
        {
            attribute + "false";
45         }
    }
}

ATTRIBUTE PIPE_DELAY
50 {
    <DIV>
    {
        attribute + "1";
    }
}

```

```

    }

ATTRIBUTE COMBINATIONAL
{
5      <DIV>
        {
            attribute + "false";
        }
    }

10     ATTRIBUTE NUM_STATES
        {
            <DIV>
                {
15                 attribute + "2";
                }
        }

20     ATTRIBUTE INTERFACE
        {
            <DIV>
                {
                    int fract;
                    int start_position_quot;
                    int end_position_remd;
25                 int mid_position_result;
                    fract = out1->fractional_bits();
                    start_position_quot = 15 - fract;
                    end_position_remd = 15 - fract;
30                 mid_position_result = fract - 1;

                    attribute + "state1: {" ;
                    attribute + component_name + "_dividend = " + in1->name( ) + "
; \n";
35                 attribute + component_name + "_divisor = " + in2->name( ) + "
; \n";
                    attribute + component_name + "_latency = 0 ; \n";
                    attribute + " goto state2 ; \n";
                    attribute + "}\n";
40                 attribute + "state2: {" ;
                    attribute + " if ( " + component_name + "_latency " + " >34
)\n";
                    attribute + "          {      if ( " + component_name + "_remd < 0
)\n";
45                 attribute + "                      {  " + out1->name( ) + "(15:" +
fract + ") = " +
                    component_name + "_quot - 1 ;\n";
                    attribute + "                      }      \n";
                    attribute + "                      else \n";
50                 attribute + "                      {  " + out1->name( ) + "(15:" +
fract + ") = " +
                    component_name + "_quot;\n";
                    attribute + "                      } \n";

```

```

        attribute + "          " + out1->name( ) + "( " +
mid_position_result + ":0 ) ="
        + component_name + "_remd ( 14:" +
end_position_remd +
5  " ) ;\n";
        attribute + "          goto NEXTSTATE; \n";
        attribute + "          } \n";
        attribute + " else \n";
        attribute + "          { " + component_name + "_latency = " +
10 component_name + "_latency + 1 ;\n";
        attribute + "          } \n";
        attribute + " }\n";

    }
15 }
}

```

```

20 ////////////////////////////////////////////////////////////////////
// Functionality and Architecture description for the XC2V250 //
// Embedded Multipliers. There are 24 multipliers arranged in 4 //
// columns with 6 multipliers per column. The location properties //
// use the following form : LOC = MULT18X18_X#Y# //
25 ////////////////////////////////////////////////////////////////////

```

```

RESOURCEDEF MULT18X18
{

```

```

30 ////////////////////////////////////////////////////////////////////
// A[15:0] : Input "in1" to multiplier //
// B[15:0] : Input "in2" to multiplier //
// P[15:0] : Output "out1" to multiplier //
////////////////////////////////////////////////////////////////////

```

```

35 ////////////////////////////////////////////////////////////////////
// Embedded multipliers //
////////////////////////////////////////////////////////////////////

```

```

40 FUNCTIONALITY MULT;

```

```

////////////////////////////////////////////////////////////////////
// This number of Embedded multiplier //
// resources that are used/instantiated //
45 // by the compiler can be controlled by //
// the user. This attribute informs the //
// compiler that this is user //
// controllable resource. //
////////////////////////////////////////////////////////////////////

```

```

50 ATTRIBUTE USERCONTROL
{
    attribute + "true";
}

```



```

    }
}

5  //////////////////////////////////////////////////
// The number of FSM states to wait for //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////////////

10 ATTRIBUTE PIPE_DELAY
    {
        <MULT>
        {
15             attribute + "1";
        }
    }

//////////////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////////////

20 ATTRIBUTE COMBINATIONAL
    {
        <MULT>
        {
25             attribute + "true";
        }
    }

30 //////////////////////////////////////////////////
// The number of FSM states that the //
// interface code spans. This is required//
// so that the MIF Interface Parser //
35 // assigns the FSM State numbers properly//
//////////////////////////////////////////////////

ATTRIBUTE NUM_STATES
    {
40         <MULT>
        {
            attribute + "1";
        }
45     }

//////////////////////////////////////////////////
// This resource can be used for input //
// bits less than 18. The interface code //
50 // has to be generated based on the input//
// bitwidths and the signtype          //
//////////////////////////////////////////////////

```

```

ATTRIBUTE INTERFACE
{
    <MULT>
    {
5         int bitval1;
          int bitval2;
          int bitval;
          int outbitval;
          bitval1 = in1->bitwidth();
10         bitval2 = in2->bitwidth();
          if(bitval1 < bitval2)
          {
              bitval = bitval2;
          }
15         else
          {
              bitval = bitval1;
          }
          outbitval = 2 * bitval;
20
          attribute + "state1: {";
          if(bitval <= 4)
          {
              if(in1->is_signed() == true)
25                 {
                     attribute + component_name + "_A(17:8) =
0;\n";
                     attribute + component_name + "_A(7:0) = " +
                        in1->name() + ";\n";
30                 }
              else
              {
                     attribute + component_name + "_A(17:4) =
0;\n";
                     attribute + component_name + "_A(3:0) = " +
                        in1->name() + ";\n";
35                 }
              if(in2->is_signed() == true)
40                 {
                     attribute + component_name + "_B(17:8) =
0;\n";
                     attribute + component_name + "_B(7:0) = " +
                        in2->name() + ";\n";
45                 }
              else
              {
                     attribute + component_name + "_B(17:4) =
0;\n";
                     attribute + component_name + "_B(3:0) = " +
                        in2->name() + ";\n";
50                 }
              }
          }
    }
}

```

```

5      if(bitval <= 8 && bitval > 4)
        {
            if(in1->is_signed() == true)
            {
                attribute + component_name + "_A(17:16) =
0;\n";
                attribute + component_name + "_A(15:0) = " +
in1->name() + ";\n";
            }
10         else
            {
                attribute + component_name + "_A(17:8) =
0;\n";
                attribute + component_name + "_A(7:0) = " +
15         in1->name() + ";\n";
            }

            if(in2->is_signed() == true)
            {
20         attribute + component_name + "_B(17:16) =
0;\n";
                attribute + component_name + "_B(15:0) = " +
in2->name() + ";\n";
            }
25         else
            {
                attribute + component_name + "_B(17:8) =
0;\n";
                attribute + component_name + "_B(7:0) = " +
30         in2->name() + ";\n";
            }
        }

35     if(bitval <= 18 && bitval > 8)
        {
            if(in1->is_signed() == true)
            {
                attribute + component_name + "_A(17:0) = " +
in1->name() + ";\n";
40         }
            else
            {
                attribute + component_name + "_A(17) = 0;\n";
                attribute + component_name + "_A(16:0) = " +
45         in1->name() + ";\n";
            }

            if(in2->is_signed() == true)
            {
50         attribute + component_name + "_B(17:0) = " +
in2->name() + ";\n";
            }
            else
            {

```

```

        attribute + component_name + "_B(17) = 0;\n";
        attribute + component_name + "_B(16:0) = " +
            in2->name() + ";\n";
    }
5      }
      //////////////////////////////////////
      // Assign the output signal          //
      //////////////////////////////////////

10     outbitval = outbitval-1;
        attribute + out1->name() + " = " +
            component_name + "_P(" + outbitval + ": 0) ;
\n";
        attribute + "goto NEXTSTATE ;\n";
15     attribute + "};";
    }
}

20

////////////////////////////////////
// Functionality and Architecture description for the XC2V250 //
// Embedded Synchronous Multipliers. These are RDL virtual //
25 // resources which use the MULT18X18 //
////////////////////////////////////

RESOURCEDEF MULT18X18S
{
30
    //////////////////////////////////////
    // A[15:0] : Input "in1" to multiplier //
    // B[15:0] : Input "in2" to multiplier //
    // P[15:0] : Output "out1" to multiplier //
35    //////////////////////////////////////

    //////////////////////////////////////
    // Embedded multipliers //
    //////////////////////////////////////
40

    FUNCTIONALITY MULT;

    ATTRIBUTE INSTANTIATION
    {
45
        //////////////////////////////////////
        // component_name is the specific Embedded //
        // Multiplier that needs to be instantiated //
        //////////////////////////////////////

50
        attribute +
            "input signed fixed[18,0] " + component_name +
            "_A;\n" +
            "input signed fixed[18,0] " + component_name +
            "_B;\n" +

```

```

        "output signed fixed[36,0] " + component_name +
        "_P;\n";

```

```

        attribute +
5      "instantiate MULT18X18S : " + component_name +
        "(" +
            "A = " + component_name + "_A," +
            "B = " + component_name + "_B," +
            "C = " + clock_name + "," +
10      "CE = " + "1," +
            "R = " + reset_name + "," +
            "P = " + component_name + "_P" +
        ");";

```

```

    }

15  //////////////////////////////////////
    // This attribute returns true if the //
    // MULT18X18S can be used to perform the //
    // multiplication operation on the inputs//
20  // The embedded multiplier is used for //
    // signed multiplication with the MSB as //
    // the sign bit. Hence, we can have a //
    // maximum of 17X17 unsigned mults //
    //////////////////////////////////////

```

```

25  ATTRIBUTE CAN_DO
    {
        <MULT>
        {
30      if((((in1->is_signed() == true && in1->bitwidth() <
19) &&
            (in2->is_signed() == true && in2->bitwidth() <
19)) ||
            ((in1->is_unsigned() == true && in1->bitwidth()
35  < 18) &&
            (in2->is_unsigned() == true && in2->bitwidth()
< 18)))
        {
            attribute + "true";
40      }
        else
        {
            attribute + "false";
        }
45      }
    }

```

```

    //////////////////////////////////////
    // The number of FSM states to wait for //
50  // before this same resource can be used //
    // for further computations. //
    //////////////////////////////////////

```

```

    ATTRIBUTE PIPE_DELAY

```

```

        {
            <MULT>
            {
                attribute + "1";
5           }
        }

////////////////////////////////////
// The embedded multipliers are supposed //
10 // to be used as combinatorial resources //
////////////////////////////////////

ATTRIBUTE COMBINATIONAL
{
15     <MULT>
        {
            attribute + "false";
        }
20     }

////////////////////////////////////
// The number of FSM states that the      //
// interface code spans. This is required//
// so that the MIF Interface Parser      //
25 // assigns the FSM State numbers properly//
////////////////////////////////////

ATTRIBUTE NUM_STATES
{
30     <MULT>
        {
            attribute + "2";
        }
35     }

////////////////////////////////////
// This resource can be used for input    //
// bits less than 18. The interface code //
40 // has to be generated based on the input//
// bitwidths and the signtype           //
////////////////////////////////////

45 ATTRIBUTE INTERFACE
{
    <MULT>
    {

50         int bitval1;
            int bitval2;
            int bitval;
            int outbitval;
            bitval1 = in1->bitwidth();

```

```

        bitval2 = in2->bitwidth();
        if(bitval1 < bitval2)
        {
            bitval = bitval2;
        }
        else
        {
            bitval = bitval1;
        }
        outbitval = 2 * bitval;

        attribute + "state1: {";
        if(bitval <= 18)
        {
            if(in1->is_signed() == true)
            {
                attribute + component_name + "_A(17:0) = " +
                    in1->name() + ";\n";
            }
            else
            {
                attribute + component_name + "_A(17) = 0;\n";
                attribute + component_name + "_A(16:0) = " +
                    in1->name() + ";\n";
            }

            if(in2->is_signed() == true)
            {
                attribute + component_name + "_B(17:0) = " +
                    in2->name() + ";\n";
            }
            else
            {
                attribute + component_name + "_B(17) = 0;\n";
                attribute + component_name + "_B(16:0) = " +
                    in2->name() + ";\n";
            }
        }
        attribute + "goto state2 ;\n";
        attribute + "}";
        //////////////////////////////////////
        // Assign the output signal
        //////////////////////////////////////

        attribute + "state2 : { \n";
        outbitval = outbitval-1;
        attribute + out1->name() + " = " +
            component_name + "_P(" + outbitval + ": 0) ;
        \n";

        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }
}

```

```

5  ////////////////////////////////////////////////////////////////////
  // Functionality and Architecture description for the XC2V250 Block //
  // SelectRAM. This device has 24 SelectRAM memory blocks with storage //
  // for 432 Kb of data. The location properties use the following //
  // form : LOC = RAMB16_X#Y#. //
  ////////////////////////////////////////////////////////////////////
10 FUNCTIONALITYDEF READ_MEM_ACCESS
   {
       OUTPUT q;
   }
15 FUNCTIONALITYDEF WRITE_MEM_ACCESS
   {
       INPUT q;
   }
20
  ////////////////////////////////////////////////////////////////////
  // Read/Write Port A for true dual port memory //
  ////////////////////////////////////////////////////////////////////
25 RESOURCEDEF RAM_PORT_A_S1_S1 {

   ////////////////////////////////////////////////////////////////////
30   // True Dual ported RAM //
   ////////////////////////////////////////////////////////////////////

   FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

35   ////////////////////////////////////////////////////////////////////
   // The number of FSM states to wait for //
   // before this same resource can be used //
   // for further computations. //
40   ////////////////////////////////////////////////////////////////////

   ATTRIBUTE PIPE_DELAY
   {
75       attribute + "1";
   }

   ////////////////////////////////////////////////////////////////////
50   // The embedded multipliers are supposed //
   // to be used as combinatorial resources //
   ////////////////////////////////////////////////////////////////////

   ATTRIBUTE COMBINATIONAL
   {

```



```

        <READ_MEM_ACCESS>
        {
            attribute + "false";
        }
5      <WRITE_MEM_ACCESS>
        {
            attribute + "false";
        }
    }

10     //////////////////////////////////////
    // This attribute returns true if the    //
    // input and output data widths are      //
    // 1 bit wide                           //
15     //////////////////////////////////////

    ATTRIBUTE CAN_DO
    {
        <READ_MEM_ACCESS>
20        {
            attribute + "true";
        }
        <WRITE_MEM_ACCESS>
25        {
            attribute + "true";
        }
    }

30     //////////////////////////////////////
    // The number of FSM states that the    //
    // interface code spans. This is required//
    // so that the MIF Interface Parser      //
    // assigns the FSM State numbers properly//
35     //////////////////////////////////////

    ATTRIBUTE NUM_STATES
    {
        <READ_MEM_ACCESS>
40        {
            attribute + "3";
        }
        <WRITE_MEM_ACCESS>
45        {
            attribute + "2";
        }
    }

50     //////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs            //
    // //////////////////////////////////////

    ATTRIBUTE INTERFACE

```

```

{
    <READ_MEM_ACCESS>
    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
            component_name + "_WEA = 0; \n" +
            component_name + "_ENA = 1; \n" +
            component_name + "_DIA = 0; \n" +
            component_name + "_DIPA = 0; \n" +
            component_name + "_ADDRA = " +
            mem_address + "; \n";

        attribute + "goto state2 ;\n";
        attribute + "}\"";

        attribute + "state2 : {";
        attribute + "goto state3 ;\n";
        attribute + "}\"";

        attribute + "state3 : {";
        attribute + q->name() + "=" + component_name +
            "_DOA" + "; \n";
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}\"";
    }

    <WRITE_MEM_ACCESS>
    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
            component_name + "_WEA = 1; \n" +
            component_name + "_ENA = 1; \n" +
            component_name + "_ADDRA = " +
            mem_address + "; \n";
        attribute + "goto state2 ;\n";
        attribute + "}\"";

        attribute + "state2 : {";
        attribute + component_name + "_DIA" + "=" +
            q->name() + "; \n";
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}\"";
    }
}

////////////////////////////////////
//  Read/Write Port A for true dual port memory  //
////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S2_S2 {

```

```

5  //////////////////////////////////////
   // True Dual ported RAM //
   //////////////////////////////////////

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

10  //////////////////////////////////////
   // The number of FSM states to wait for //
   // before this same resource can be used //
   // for further computations. //
   //////////////////////////////////////

15  ATTRIBUTE PIPE_DELAY
   {
       attribute + "1";
   }

20  //////////////////////////////////////
   // The embedded multipliers are supposed //
   // to be used as combinatorial resources //
   //////////////////////////////////////

25  ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
       {
30         attribute + "false";
       }
       <WRITE_MEM_ACCESS>
       {
35         attribute + "false";
       }
   }

40  //////////////////////////////////////
   // This attribute returns true if the //
   // input and output data widths are //
   // 1 bit wide //
   //////////////////////////////////////

45  ATTRIBUTE CAN_DO
   {
       <READ_MEM_ACCESS>
       {
           attribute + "true";
       }
50       <WRITE_MEM_ACCESS>
       {
           attribute + "true";
       }
   }

```

```

5 //////////////////////////////////////////////////
// The number of FSM states that the //
// interface code spans. This is required//
// so that the MIF Interface Parser //
// assigns the FSM State numbers properly//
//////////////////////////////////////////////////

10 ATTRIBUTE NUM_STATES
    {
        <READ_MEM_ACCESS>
        {
            attribute + "3";
15        }
        <WRITE_MEM_ACCESS>
        {
            attribute + "2";
20        }
    }

25 //////////////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs //
//////////////////////////////////////////////////

ATTRIBUTE INTERFACE
    {
30        <READ_MEM_ACCESS>
        {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
35                component_name + "_WEA = 0; \n" +
                component_name + "_ENA = 1; \n" +
                component_name + "_DIA = 0; \n" +
                component_name + "_DIPA = 0; \n" +
                component_name + "_ADDRA = " +
40                mem_address + "; \n";

            attribute + "goto state2 ;\n";
            attribute + "}";

45            attribute + "state2 : {" +
                attribute + "goto state3 ;\n";
            attribute + "}";

            attribute + "state3 : {" +
50            attribute + q->name() + "=" + component_name +
                "_DOA" + "; \n";
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}";
        }
    }

```

```

        <WRITE_MEM_ACCESS>
        {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
                component_name + "_WEA = 1; \n" +
                component_name + "_ENA = 1; \n" +
                component_name + "_ADDRA = " +
            mem_address + "; \n";
            attribute + "goto state2 ;\n";
            attribute + "}";

            attribute + "state2 : {" +
            attribute + component_name + "_DIA" + "=" +
                q->name() + "; \n";
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}";
        }
    }

    //////////////////////////////////////
    // Read/Write Port A for true dual port memory //
    //////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S4_S4 {

    //////////////////////////////////////
    // True Dual ported RAM //
    //////////////////////////////////////

    FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

    //////////////////////////////////////
    // The number of FSM states to wait for //
    // before this same resource can be used //
    // for further computations. //
    //////////////////////////////////////

    ATTRIBUTE PIPE_DELAY
    {
        attribute + "1";
    }

    //////////////////////////////////////
    // The embedded multipliers are supposed //
    // to be used as combinatorial resources //
    //////////////////////////////////////

    ATTRIBUTE COMBINATIONAL

```

```

    {
        <READ_MEM_ACCESS>
        {
            attribute + "false";
5         }
        <WRITE_MEM_ACCESS>
        {
            attribute + "false";
10        }
    }

    //////////////////////////////////////
    // This attribute returns true if the    //
    // input and output data widths are      //
15    // 1 bit wide                          //
    //////////////////////////////////////

    ATTRIBUTE CAN_DO
    {
20        <READ_MEM_ACCESS>
        {
            attribute + "true";
        }
        <WRITE_MEM_ACCESS>
25        {
            attribute + "true";
        }
    }

30    //////////////////////////////////////
    // The number of FSM states that the    //
    // interface code spans. This is required//
    // so that the MIF Interface Parser      //
35    // assigns the FSM State numbers properly//
    //////////////////////////////////////

    ATTRIBUTE NUM_STATES
    {
40        <READ_MEM_ACCESS>
        {
            attribute + "3";
        }
        <WRITE_MEM_ACCESS>
45        {
            attribute + "2";
        }
    }

50    //////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs            //
    //////////////////////////////////////

```

ATTRIBUTE INTERFACE

```

{
    <READ_MEM_ACCESS>
    {
        5      int bitval;
              bitval = q->bitwidth();
              attribute + "state1 : {" +
                  component_name + "_WEA = 0; \n" +
                  component_name + "_ENA = 1; \n" +
10             component_name + "_DIA = 0; \n" +
                  component_name + "_DIPA = 0; \n" +
                  component_name + "_ADDRA = " +
                      mem_address + "; \n";

              attribute + "goto state2 ;\n";
              attribute + "}";

              attribute + "state2 : {";
              attribute + "goto state3 ;\n";
20             attribute + "}";

              attribute + "state3 : {";
              attribute + q->name() + "=" + component_name +
                  "_DOA" + "; \n";
25             attribute + "goto NEXTSTATE ;\n";
              attribute + "}";
        }

        <WRITE_MEM_ACCESS>
        {
            30      int bitval;
                  bitval = q->bitwidth();
                  attribute + "state1 : {" +
                      component_name + "_WEA = 1; \n" +
                      component_name + "_ENA = 1; \n" +
35                     component_name + "_ADDRA = " +
                        mem_address + "; \n";
                  attribute + "goto state2 ;\n";
                  attribute + "}";

            40      attribute + "state2 : {";
                  attribute + component_name + "_DIA" + "=" +
                      q->name() + "; \n";
                  attribute + "goto NEXTSTATE ;\n";
            45      attribute + "}";
        }
    }

    }

    50  ////////////////////////////////////////////////////////////////////
        //  Read/Write Port A for true dual port memory              //
        ////////////////////////////////////////////////////////////////////

    RESOURCEDEF RAM_PORT_A_S9_S9 {

```

```

5  //////////////////////////////////////
   // True Dual ported RAM           //
   //////////////////////////////////////

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

10  //////////////////////////////////////
   // The number of FSM states to wait for //
   // before this same resource can be used //
   // for further computations.           //
   //////////////////////////////////////

15

ATTRIBUTE PIPE_DELAY
{
20     attribute + "1";
}

   //////////////////////////////////////
   // The embedded multipliers are supposed //
   // to be used as combinatorial resources //
   //////////////////////////////////////

25

ATTRIBUTE COMBINATIONAL
{
30     <READ_MEM_ACCESS>
        {
            attribute + "false";
        }
        <WRITE_MEM_ACCESS>
        {
35             attribute + "false";
        }
}

   //////////////////////////////////////
40  // This attribute returns true if the //
   // input and output data widths are    //
   // 1 bit wide                          //
   //////////////////////////////////////

45  ATTRIBUTE CAN_DO
{
    <READ_MEM_ACCESS>
    {
50         attribute + "true";
    }
    <WRITE_MEM_ACCESS>
    {
        attribute + "true";
    }
}

```


Sample	Temperature (°C)	Time (h)	Yield (%)	Color	Odor	Flavor	Texture	Stability	Notes
1	100	1	85	Light yellow	Weak	Light	Soft	Good	Control
2	120	1	75	Yellow	Medium	Medium	Medium	Good	Increased yield and color
3	140	1	65	Dark yellow	Strong	Strong	Medium	Good	Increased yield and color, strong odor
4	160	1	55	Brown	Very strong	Very strong	Hard	Good	Increased yield and color, very strong odor, hard texture
5	180	1	45	Dark brown	Extremely strong	Extremely strong	Very hard	Good	Increased yield and color, extremely strong odor, very hard texture
6	200	1	35	Black	Not detectable	Not detectable	Brittle	Good	Increased yield and color, not detectable odor, brittle texture
7	220	1	25	Black	Not detectable	Not detectable	Brittle	Good	Increased yield and color, not detectable odor, brittle texture
8	240	1	15	Black	Not detectable	Not detectable	Brittle	Good	Increased yield and color, not detectable odor, brittle texture
9	260	1	5	Black	Not detectable	Not detectable	Brittle	Good	Increased yield and color, not detectable odor, brittle texture
10	280	1	0	Black	Not detectable	Not detectable	Brittle	Good	Increased yield and color, not detectable odor, brittle texture

5


```

}

////////////////////////////////////
// Read/Write Port A for true dual port memory //
5  //////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S18_S18 {

10  //////////////////////////////////////
    // True Dual ported RAM //
    //////////////////////////////////////

15  FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

    //////////////////////////////////////
    // The number of FSM states to wait for //
    // before this same resource can be used //
20  // for further computations. //
    //////////////////////////////////////

    ATTRIBUTE PIPE_DELAY
25  {
        attribute + "1";
    }

    //////////////////////////////////////
    // The embedded multipliers are supposed //
    // to be used as combinatorial resources //
30  //////////////////////////////////////

    ATTRIBUTE COMBINATIONAL
35  {
        <READ_MEM_ACCESS>
        {
            attribute + "false";
        }
        <WRITE_MEM_ACCESS>
40  {
            attribute + "false";
        }
    }

45  //////////////////////////////////////
    // This attribute returns true if the //
    // input and output data widths are //
    // 1 bit wide //
50  //////////////////////////////////////

    ATTRIBUTE CAN_DO
    {
        <READ_MEM_ACCESS>
    }

```

```

        {
            attribute + "true";
        }
<WRITE_MEM_ACCESS>
5      {
        attribute + "true";
    }
}

10
////////////////////////////////////
// The number of FSM states that the //
// interface code spans. This is required//
// so that the MIF Interface Parser //
15 // assigns the FSM State numbers properly//
////////////////////////////////////

ATTRIBUTE NUM_STATES
{
20     <READ_MEM_ACCESS>
        {
            attribute + "3";
        }
    <WRITE_MEM_ACCESS>
25     {
        attribute + "2";
    }
}

30
////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs //
////////////////////////////////////

35 ATTRIBUTE INTERFACE
{
    <READ_MEM_ACCESS>
    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
            component_name + "_WEA = 0; \n" +
            component_name + "_ENA = 1; \n" +
            component_name + "_DIA = 0; \n" +
45         component_name + "_DIPA = 0; \n" +
            component_name + "_ADDRA = " +
            mem_address + "; \n";

        attribute + "goto state2 ;\n";
50     attribute + "}";

        attribute + "state2 : {";
        attribute + "goto state3 ;\n";
        attribute + "}";
    }
}

```



```

        }
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }
5      }

// Read/Write Port A for true dual port memory
10     //
RESOURCEDEF RAM_PORT_A_S36_S36 {

15     // True Dual ported RAM
    //
    FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

20     // The number of FSM states to wait for
    // before this same resource can be used
    // for further computations.
25     //
    ATTRIBUTE PIPE_DELAY
30     {
        attribute + "1";
    }

    // The embedded multipliers are supposed
35     // to be used as combinatorial resources
    //
    ATTRIBUTE COMBINATIONAL
40     {
        <READ_MEM_ACCESS>
        {
            attribute + "false";
        }
        <WRITE_MEM_ACCESS>
45     {
            attribute + "false";
        }
    }

50     // This attribute returns true if the
    // input and output data widths are
    // 1 bit wide

```

```

////////////////////////////////////
ATTRIBUTE CAN_DO
5      {
        <READ_MEM_ACCESS>
        {
            attribute + "true";
        }
        <WRITE_MEM_ACCESS>
10     {
            attribute + "true";
        }
    }

15
////////////////////////////////////
// The number of FSM states that the //
// interface code spans. This is required//
// so that the MIF Interface Parser //
20 // assigns the FSM State numbers properly//
////////////////////////////////////

ATTRIBUTE NUM_STATES
25     {
        <READ_MEM_ACCESS>
        {
            attribute + "3";
        }
        <WRITE_MEM_ACCESS>
30     {
            attribute + "2";
        }
    }

35
////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs //
////////////////////////////////////

40 ATTRIBUTE INTERFACE
    {
        <READ_MEM_ACCESS>
        {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
            component_name + "_WEA = 0; \n" +
            component_name + "_ENA = 1; \n" +
            component_name + "_DIA = 0; \n" +
50     component_name + "_DIPA = 0; \n" +
            component_name + "_ADDRA = " +
            mem_address + "; \n";

            attribute + "goto state2 ;\n";
        }
    }

```

```

attribute + "}";

attribute + "state2 : {";
attribute + "goto state3 ;\n";
5 attribute + "}";

attribute + "state3 : {";
if(q->bitwidth() <= 32)
{
10 attribute + q->name() + "=" +
component_name +
        "_DOA" + "; \n";
}
else
15 {
    bitval = bitval - 1;
    attribute + q->name() + "(" + bitval +
        ":32" +
        component_name + "_DOPA" + "; \n";
20 attribute + q->name() + "(31:0) = " +
        component_name + "_DOA" + "; \n";
    }
    attribute + "goto NEXTSTATE ;\n";
    attribute + "}";
25 }

<WRITE_MEM_ACCESS>
{
    int bitval;
    bitval = q->bitwidth();
30 attribute + "state1 : {" +
        component_name + "_WEA = 1; \n" +
        component_name + "_ENA = 1; \n" +
        component_name + "_ADDRA = " +
        mem_address + "; \n";
35 attribute + "goto state2 ;\n";
    attribute + "}";

    attribute + "state2 : {";
    if(q->bitwidth() <= 32)
    {
40 attribute + component_name + "_DIPA = 0;
        \n";
        attribute + component_name + "_DIA" +
        "=" +
45 q->name() + "; \n";
    }
    else
    {
50 bitval = bitval - 1;
        attribute + component_name + "_DIPA = "
        +
        q->name() + "(" + bitval + ")" + ";
        \n";

```



```

        bitval = bitval - 1;
        attribute + component_name + "_DIA" +

```

```

"=" +

```

```

5          q->name() + "(" + bitval +
          ":0)" + ";\n";

```

```

          }
          attribute + "goto NEXTSTATE ;\n";
          attribute + "};
        }

```

```

10      }
    }

```

```

15  //////////////////////////////////////
  //  Read/Write Port B for true dual port memory  //
  //////////////////////////////////////

```

```

20  RESOURCEDEF RAM_PORT_B_S1_S1 {

```

```

        //////////////////////////////////////
        //  True Dual ported RAM  //
        //////////////////////////////////////

```

```

25  FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

```

```

30  //////////////////////////////////////
  //  The number of FSM states to wait for  //
  //  before this same resource can be used  //
  //  for further computations.  //
  //////////////////////////////////////

```

```

35
  ATTRIBUTE PIPE_DELAY
  {
    attribute + "1";
  }
40

```

```

        //////////////////////////////////////
        //  The embedded multipliers are supposed  //
        //  to be used as combinatorial resources  //
        //////////////////////////////////////

```

```

45
  ATTRIBUTE COMBINATIONAL
  {

```

```

50      <READ_MEM_ACCESS>
      {
        attribute + "false";
      }
      <WRITE_MEM_ACCESS>
      {

```

```

        attribute + "false";
    }
}

5  //////////////////////////////////////
   // This attribute returns true if the //
   // input and output data widths are   //
   // 1 bit wide                          //
   //////////////////////////////////////
10
   ATTRIBUTE CAN_DO
   {
       <READ_MEM_ACCESS>
       {
15           attribute + "true";
       }
       <WRITE_MEM_ACCESS>
       {
20           attribute + "true";
       }
   }

   //////////////////////////////////////
25  // The number of FSM states that the //
   // interface code spans. This is required//
   // so that the MIF Interface Parser    //
   // assigns the FSM State numbers properly//
   //////////////////////////////////////
30
   ATTRIBUTE NUM_STATES
   {
       <READ_MEM_ACCESS>
       {
35           attribute + "3";
       }
       <WRITE_MEM_ACCESS>
       {
40           attribute + "2";
       }
   }

   //////////////////////////////////////
45  // The interface for reading and writing //
   // from the Block SelectRAMs           //
   //////////////////////////////////////

   ATTRIBUTE INTERFACE
   {
50       <READ_MEM_ACCESS>
       {
           int bitval;
           bitval = q->bitwidth();
           attribute + "state1 : {" +

```



```

// assigns the FSM State numbers properly//
////////////////////////////////////////

5      ATTRIBUTE NUM_STATES
      {
          <READ_MEM_ACCESS>
          {
              attribute + "3";
          }
10      <WRITE_MEM_ACCESS>
          {
              attribute + "2";
          }
      }

15

////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs           //
20  //////////////////////////////////

ATTRIBUTE INTERFACE
{
    <READ_MEM_ACCESS>
25  {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
            component_name + "_WEB = 0; \n" +
30        component_name + "_ENB = 1; \n" +
            component_name + "_DIB = 0; \n" +
            component_name + "_DIPB = 0; \n" +
            component_name + "_ADDRB = " +
            mem_address + "; \n";

35        attribute + "goto state2 ;\n";
        attribute + "}";

        attribute + "state2 : {";
        attribute + "goto state3 ;\n";
40        attribute + "}";

        attribute + "state3 : {";
        attribute + q->name() + "=" + component_name +
45        "_DOB" + "; \n";
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }

50    <WRITE_MEM_ACCESS>
    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +

```

```

        component_name + "_WEB = 1; \n" +
        component_name + "_ENB = 1; \n" +
        component_name + "_ADDRB = " +
        mem_address + "; \n";
5      attribute + "goto state2 ;\n";
      attribute + "}";

      attribute + "state2 : {";
      attribute + component_name + "_DIB" + "=" +
10      q->name() + "; \n";
      attribute + "goto NEXTSTATE ;\n";
      attribute + "}";
    }
  }
15 }

////////////////////////////////////
// Read/Write Port B for true dual port memory //
////////////////////////////////////

20 RESOURCEDEF RAM_PORT_B_S4_S4 {

    //////////////////////////////////
    // True Dual ported RAM //
    //////////////////////////////////

    FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

30

    //////////////////////////////////
    // The number of FSM states to wait for //
    // before this same resource can be used //
    // for further computations. //
    //////////////////////////////////
35

    ATTRIBUTE PIPE_DELAY
    {
40      attribute + "1";
    }

    //////////////////////////////////
    // The embedded multipliers are supposed //
    // to be used as combinatorial resources //
    //////////////////////////////////
45

    ATTRIBUTE COMBINATIONAL
    {
50      <READ_MEM_ACCESS>
      {
        attribute + "false";
      }
      <WRITE_MEM_ACCESS>

```

```

        {
            attribute + "false";
        }
    }

5
    //////////////////////////////////////
    // This attribute returns true if the    //
    // input and output data widths are      //
    // 1 bit wide                           //
10    //////////////////////////////////////

    ATTRIBUTE CAN_DO
    {
        <READ_MEM_ACCESS>
15        {
            attribute + "true";
        }
        <WRITE_MEM_ACCESS>
20        {
            attribute + "true";
        }
    }

25
    //////////////////////////////////////
    // The number of FSM states that the    //
    // interface code spans. This is required//
    // so that the MIF Interface Parser      //
    // assigns the FSM State numbers properly//
30    //////////////////////////////////////

    ATTRIBUTE NUM_STATES
    {
        <READ_MEM_ACCESS>
35        {
            attribute + "3";
        }
        <WRITE_MEM_ACCESS>
40        {
            attribute + "2";
        }
    }

45
    //////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs            //
    //////////////////////////////////////

    ATTRIBUTE INTERFACE
50    {
        <READ_MEM_ACCESS>
        {
            int bitval;
            bitval = q->bitwidth();

```

```

        attribute + "state1 : {" +
            component_name + "_WEB = 0; \n" +
            component_name + "_ENB = 1; \n" +
            component_name + "_DIB = 0; \n" +
            component_name + "_DIPB = 0; \n" +
            component_name + "_ADDRB = " +
                mem_address + "; \n";

        attribute + "goto state2 ;\n";
        attribute + "}";

        attribute + "state2 : {";
        attribute + "goto state3 ;\n";
        attribute + "}";

        attribute + "state3 : {";
        attribute + q->name() + "=" + component_name +
            "_DOB" + "; \n";
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }

    <WRITE_MEM_ACCESS>
    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
            component_name + "_WEB = 1; \n" +
            component_name + "_ENB = 1; \n" +
            component_name + "_ADDRB = " +
                mem_address + "; \n";
        attribute + "goto state2 ;\n";
        attribute + "}";

        attribute + "state2 : {";
        attribute + component_name + "_DIB" + "=" +
            q->name() + "; \n";
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }
}

////////////////////////////////////
45 // Read/Write Port B for true dual port memory //
////////////////////////////////////

RESOURCEDEF RAM_PORT_B_S9_S9 {

50
    //////////////////////////////////
    // True Dual ported RAM //
    //////////////////////////////////

```


FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

```

////////////////////////////////////////
// The number of FSM states to wait for //
// before this same resource can be used //
// for further computations.             //
////////////////////////////////////////

```

```

ATTRIBUTE PIPE_DELAY
{
    attribute + "1";
}

```

```

////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
////////////////////////////////////////

```

```

ATTRIBUTE COMBINATIONAL
{
    <READ_MEM_ACCESS>
    {
        attribute + "false";
    }
    <WRITE_MEM_ACCESS>
    {
        attribute + "false";
    }
}

```

```

////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
////////////////////////////////////////

```

```

ATTRIBUTE CAN_DO
{
    <READ_MEM_ACCESS>
    {
        attribute + "true";
    }
    <WRITE_MEM_ACCESS>
    {
        attribute + "true";
    }
}

```

ATTRIBUTE NUM_STATES

```

    {
        <READ_MEM_ACCESS>
        {
            attribute + "3";
5           }
        <WRITE_MEM_ACCESS>
        {
            attribute + "2";
10          }
    }

    //////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs           //
15    //////////////////////////////////////

    ATTRIBUTE INTERFACE
    {
        <READ_MEM_ACCESS>
20        {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
25                component_name + "_WEB = 0; \n" +
                component_name + "_ENB = 1; \n" +
                component_name + "_DIB = 0; \n" +
                component_name + "_DIPB = 0; \n" +
                component_name + "_ADDRB = " +
                mem_address + "; \n";

30                attribute + "goto state2 ;\n";
                attribute + "}";

                attribute + "state2 : {";
                attribute + "goto state3 ;\n";
                attribute + "}";

35                attribute + "state3 : {";

40                if(q->bitwidth() <= 8)
                {
                    attribute + q->name() + "=" +
                    component_name +
                    "_DOB" + "; \n";
45                }
                else
                {
                    bitval = bitval - 1;
                    attribute + q->name() + "(" + bitval +
50                    ":8" +
                    component_name + "_DOPB" + "; \n";
                    attribute + q->name() + "(7:0) = " +
                    component_name + "_DOB" + "; \n";
                }
            }
        }
    }

```



```

////////////////////////////////////
FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

```

```

////////////////////////////////////
// The number of FSM states to wait for //
// before this same resource can be used //
// for further computations. //
////////////////////////////////////

```

```

ATTRIBUTE PIPE_DELAY
{
    attribute + "1";
}

```

```

////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
////////////////////////////////////

```

```

ATTRIBUTE COMBINATIONAL
{
    <READ_MEM_ACCESS>
    {
        attribute + "false";
    }
    <WRITE_MEM_ACCESS>
    {
        attribute + "false";
    }
}

```

```

////////////////////////////////////
// This attribute returns true if the //
// input and output data widths are //
// 1 bit wide //
////////////////////////////////////

```

```

ATTRIBUTE CAN_DO
{
    <READ_MEM_ACCESS>
    {
        attribute + "true";
    }
    <WRITE_MEM_ACCESS>
    {
        attribute + "true";
    }
}

```

```

ATTRIBUTE NUM_STATES

```

```

    {
        <READ_MEM_ACCESS>
        {
            attribute + "3";
5           }
        <WRITE_MEM_ACCESS>
        {
            attribute + "2";
10          }
    }

    //////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs           //
15    //////////////////////////////////////

    ATTRIBUTE INTERFACE
    {
        <READ_MEM_ACCESS>
20        {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
25                component_name + "_WEB = 0; \n" +
                component_name + "_ENB = 1; \n" +
                component_name + "_DIB = 0; \n" +
                component_name + "_DIPB = 0; \n" +
                component_name + "_ADDRB = " +
                mem_address + "; \n";

30                attribute + "goto state2 ; \n";
                attribute + "}";

                attribute + "state2 : {";
35                attribute + "goto state3 ; \n";
                attribute + "}";

                attribute + "state3 : {";
                if(q->bitwidth() <= 16)
40                {
                    attribute + q->name() + "=" +
component_name +
                    "_DOB" + "; \n";
                }
45                else
                {
                    bitval = bitval - 1;
                    attribute + q->name() + "(" + bitval +
50                    " : 16" +
                    component_name + "_DOPB" + "; \n";
                    attribute + q->name() + "(15:0) = " +
                    component_name + "_DOB" + "; \n";
                }
                attribute + "goto NEXTSTATE ; \n";

```

```

        attribute + "}";
    }

    <WRITE_MEM_ACCESS>
5      {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
10          component_name + "_WEB = 1; \n" +
            component_name + "_ENB = 1; \n" +
            component_name + "_ADDRB = " +
            mem_address + "; \n";
        attribute + "goto state2 ;\n";
        attribute + "}";

15          attribute + "state2 : {";
            if(q->bitwidth() <= 16)
                {
20                  attribute + component_name + "_DIPB = 0;

                    attribute + component_name + "_DIB" +
                    "=" +
                    q->name() + "; \n";
                }
25            else
                {
                    bitval = bitval - 1;
                    attribute + component_name + "_DIPB = "
30                    +
                    q->name() + "(" + bitval + ")" + ";

                    bitval = bitval - 1;
                    attribute + component_name + "_DIB" +
                    "=" +
35                    q->name() + "(" + bitval +
                    ":0)" + "; \n";
                }
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}";
40        }
    }

    ////////////////////////////////////////
45    //  Read/Write Port B for true dual port memory  //
    ////////////////////////////////////////

    RESOURCEDEF RAM_PORT_B_S36_S36 {

50        ////////////////////////////////////////
        //  True Dual ported RAM  //
        ////////////////////////////////////////

```

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

5 ///
// The number of FSM states to wait for //
// before this same resource can be used //
// for further computations. //
////////////////////////////////////

10
ATTRIBUTE PIPE_DELAY
{
 attribute + "1";
}

15
////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
////////////////////////////////////

20
ATTRIBUTE COMBINATIONAL
{
 <READ_MEM_ACCESS>
 {
25 attribute + "false";
 }
 <WRITE_MEM_ACCESS>
 {
30 attribute + "false";
 }
}

35
////////////////////////////////////
// This attribute returns true if the //
// input and output data widths are //
// 1 bit wide //
////////////////////////////////////

40
ATTRIBUTE CAN_DO
{
 <READ_MEM_ACCESS>
 {
 attribute + "true";
 }
45 <WRITE_MEM_ACCESS>
 {
 attribute + "true";
 }
50}

////////////////////////////////////
// The number of FSM states that the //
// interface code spans. This is required//

```

// so that the MIF Interface Parser      //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

5  ATTRIBUTE NUM_STATES
    {
        <READ_MEM_ACCESS>
        {
            attribute + "3";
10  }
        <WRITE_MEM_ACCESS>
        {
            attribute + "2";
15  }
    }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs           //
20  ////////////////////////////////////////////

ATTRIBUTE INTERFACE
    {
        <READ_MEM_ACCESS>
25  {
            int bitval;
            bitval = q->bitwidth();
            attribute + "state1 : {" +
                component_name + "_WEB = 0; \n" +
30  component_name + "_ENB = 1; \n" +
                component_name + "_DIB = 0; \n" +
                component_name + "_DIPB = 0; \n" +
                component_name + "_ADDRB = " +
35  mem_address + "; \n";

            attribute + "goto state2 ;\n";
            attribute + "}";

            attribute + "state2 : {";
            attribute + "goto state3 ;\n";
40  attribute + "}";

            attribute + "state3 : {";
            if(q->bitwidth() <= 32)
            {
                attribute + q->name() + "=" +
45  component_name +
                    "_DOB" + "; \n";
            }
            else
            {
                bitval = bitval - 1;
                attribute + q->name() + "(" + bitval +
50  ":32" +

```



```

        component_name + "_DOPB" + "; \n";
        attribute + q->name() + "(31:0) = " +
        component_name + "_DOB" + "; \n";
    }
5      attribute + "goto NEXTSTATE ;\n";
      attribute + "}";
    }

    <WRITE_MEM_ACCESS>
10    {
        int bitval;
        bitval = q->bitwidth();
        attribute + "state1 : {" +
        component_name + "_WEB = 1; \n" +
15      component_name + "_ENB = 1; \n" +
        component_name + "_ADDRB = " +
        mem_address + "; \n";
        attribute + "goto state2 ;\n";
        attribute + "}";

20      attribute + "state2 : {";
        if(q->bitwidth() <= 32)
        {
            attribute + component_name + "_DIPB = 0;
25      \n";
            attribute + component_name + "_DIB" +
            "=" +
            q->name() + "; \n";
        }
30      else
        {
            bitval = bitval - 1;
            attribute + component_name + "_DIPB = "
+
35      q->name() + "(" + bitval + ")" + ";
            \n";
            bitval = bitval - 1;
            attribute + component_name + "_DIB" +
            "=" +
40      q->name() + "(" + bitval +
            ":0)" + "; \n";
        }
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
45    }
    }

50  ////////////////////////////////////////////
    // A XC2V250 Block SelectRAM which is configured to work //
    // at MODE S1_S1 with aspect ratio of both ports as      //
    // A: 16,384 x 1 bit and B: 16,384 X 1 bit                //

```

////////////////////////////////////

RESOURCEDEF RAM_MODE_S1_S1 {

5 ////////////////////////////////////
 // This is connected to Ports //
 // which has read/write func //
 ////////////////////////////////////

10 FUNCTIONALITY STORAGE;

 ////////////////////////////////////
 // A Dual Port RAM //
 ////////////////////////////////////

15 TAGS DUALPORT;

 ////////////////////////////////////
 // The number of addressable locations //
 ////////////////////////////////////

20 ATTRIBUTE MAX_LOCATIONS
 {
25 attribute + 16384;
 }

 ////////////////////////////////////
 // The width of each addressable locations //
 ////////////////////////////////////

30 ATTRIBUTE MAX_WIDTH
 {
35 attribute + 1;
 }

 ////////////////////////////////////
 // The code to instantiate the RAM //
 // in mode S1_S1 //
 ////////////////////////////////////

40 ATTRIBUTE INSTANTIATION
 {

45 ////////////////////////////////////
 // component_name is the specific RAM that //
 // needs to be accessed //
 ////////////////////////////////////

50 ////////////////////////////////////
 // These signals have to be declared so that//
 // the compiler has them in the Symbol Table//
 // before it parses in the MIF interface //
 // code from the ports. //
 ////////////////////////////////////

```

        attribute +
        "input unsigned fixed[14,0] " + component_name +
        "_ADDRB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_DIA;\n"+
        "output unsigned fixed[1,0] " + component_name +
        "_DOA;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_WEA;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_ENA;\n";

        attribute +
        "input unsigned fixed[14,0] " + component_name +
        "_ADDRB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_DIB;\n"+
        "output unsigned fixed[1,0] " + component_name +
        "_DOB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_WEB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_ENB;\n";

        attribute +
        "instantiate RAMB16_S1_S1 : " + component_name +
        "(" +
        "DIA = " + component_name + "_DIA," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOA = " + component_name + "_DOA," +
        "ENA = " + component_name + "_ENA," +
        "WEA = " + component_name + "_WEA," +
        "CLKA = " + clock_name + " , " +
        "SSRA = " + reset_name + " , " +
        "DIB = " + component_name + "_DIB," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOB = " + component_name + "_DOB," +
        "ENB = " + component_name + "_ENB," +
        "WEB = " + component_name + "_WEB," +
        "CLKB = " + clock_name + " , " +
        "SSRB = " + reset_name +
        ");";

    }

}

////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S2_S2 with aspect ratio of both ports as      //
// A: 8192 x 2 bit and B: 8192 X 2 bit                    //
////////////////////////////////////////

```

RESOURCEDEF RAM_MODE_S2_S2 {

```
5          //////////////////////////////////////
          // This is connected to Ports //
          // which has read/write func //
          //////////////////////////////////////

10         FUNCTIONALITY STORAGE;

          //////////////////////////////////////
          // A Dual Port RAM           //
          //////////////////////////////////////

15         TAGS DUALPORT;

          //////////////////////////////////////
          // The number of addressable locations //
          //////////////////////////////////////

20         ATTRIBUTE MAX_LOCATIONS
          {
            attribute + 8192;
          }

25          //////////////////////////////////////
          // The width of each addressable locations //
          //////////////////////////////////////

30         ATTRIBUTE MAX_WIDTH
          {
            attribute + 2;
          }

35          //////////////////////////////////////
          // The code to instantiate the RAM           //
          // in mode S2_S2                             //
          //////////////////////////////////////

40         ATTRIBUTE INSTANTIATION
          {
            //////////////////////////////////////
            // component_name is the specific RAM that //
            // needs to be accessed                     //
            //////////////////////////////////////

45          //////////////////////////////////////

50          // These signals have to be declared so that//
          // the compiler has them in the Symbol Table//
          // before it parses in the MIF interface    //
          // code from the ports.                    //
          //////////////////////////////////////
```



```

        "input unsigned fixed[12,0] " + component_name +
        "_ADDRB;\n"+
        "input unsigned fixed[4,0] " + component_name +
        "output unsigned fixed[4,0] " + component_name +
        "input unsigned fixed[1,0] " + component_name +
        "input unsigned fixed[1,0] " + component_name +
        attribute +
        "input unsigned fixed[12,0] " + component_name +
        "input unsigned fixed[4,0] " + component_name +
        "output unsigned fixed[4,0] " + component_name +
        "input unsigned fixed[1,0] " + component_name +
        "input unsigned fixed[1,0] " + component_name +
        attribute +
        "instantiate RAMB16_S4_S4 : " + component_name +
        "(" +
        "DIA = " + component_name + "_DIA," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOA = " + component_name + "_DOA," +
        "ENA = " + component_name + "_ENA," +
        "WEA = " + component_name + "_WEA," +
        "CLKA = " + clock_name + " , " +
        "SSRA = " + reset_name + " , " +
        "DIB = " + component_name + "_DIB," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOB = " + component_name + "_DOB," +
        "ENB = " + component_name + "_ENB," +
        "WEB = " + component_name + "_WEB," +
        "CLKB = " + clock_name + " , " +
        "SSRB = " + reset_name +
        ");";
    }

}

// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S9_S9 with aspect ratio of both ports as //
// A: 2048 x 9 bit and B: 2048 X 9 bit //
RESOURCEDEF RAM_MODE_S9_S9 {
    // This is connected to Ports //

```

```

// which has read/write func //
////////////////////////////////////

5      FUNCTIONALITY STORAGE;

      //////////////////////////////////
      // A Dual Port RAM          //
      //////////////////////////////////

10     TAGS DUALPORT;

      //////////////////////////////////
      // The number of addressable locations //
      //////////////////////////////////

15     ATTRIBUTE MAX_LOCATIONS
      {
          attribute + 2048;
      }

20     //////////////////////////////////
      // The width of each addressable locations //
      //////////////////////////////////

25     ATTRIBUTE MAX_WIDTH
      {
          attribute + 9;
      }

30     //////////////////////////////////
      // The code to instantiate the RAM          //
      // in mode S9_S9                             //
      //////////////////////////////////

35     ATTRIBUTE INSTANTIATION
      {
          //////////////////////////////////
          // component_name is the specific RAM that //
          // needs to be accessed                     //
          //////////////////////////////////

          //////////////////////////////////
          // These signals have to be declared so that//
          // the compiler has them in the Symbol Table//
          // before it parses in the MIF interface    //
          // code from the ports.                     //
          //////////////////////////////////

45     attribute +
          "input unsigned fixed[11,0] " + component_name +
          "_ADDRA;\n"+
          "input unsigned fixed[8,0] " + component_name +
          "_DIA;\n"+

```



```

        "input unsigned fixed[1,0] " + component_name +
        "_DIPA;\n"+
        "output unsigned fixed[8,0] " + component_name +
        "_DOA;\n"+
        "output unsigned fixed[1,0] " + component_name +
        "_DOPA;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_WEA;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_ENA;\n";

        attribute +
        "input unsigned fixed[11,0] " + component_name +
        "_ADDRB;\n"+
        "input unsigned fixed[8,0] " + component_name +
        "_DIB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_DIPB;\n"+
        "output unsigned fixed[8,0] " + component_name +
        "_DOB;\n"+
        "output unsigned fixed[1,0] " + component_name +
        "_DOPB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_WEB;\n"+
        "input unsigned fixed[1,0] " + component_name +
        "_ENB;\n";

        attribute +
        "instantiate RAMB16_S9_S9 : " + component_name +
        "(" +
        "DIA = " + component_name + "_DIA," +
        "DIPA = " + component_name + "_DIPA," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOA = " + component_name + "_DOA," +
        "DOPA = " + component_name + "_DOPA," +
        "ENA = " + component_name + "_ENA," +
        "WEA = " + component_name + "_WEA," +
        "CLKA = " + clock_name + " , " +
        "SSRA = " + reset_name + " , " +
        "DIB = " + component_name + "_DIB," +
        "DIPB = " + component_name + "_DIPB," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOB = " + component_name + "_DOB," +
        "DOPB = " + component_name + "_DOPB," +
        "ENB = " + component_name + "_ENB," +
        "WEB = " + component_name + "_WEB," +
        "CLKB = " + clock_name + " , " +
        "SSRB = " + reset_name +
        ");";
    }
}

```

```

5  //////////////////////////////////////
   // A XC2V250 Block SelectRAM which is configured to work //
   // at MODE S18_S18 with aspect ratio of both ports as    //
   // A: 1024 x 18 bit and B: 1024 X 18 bit                  //
   //////////////////////////////////////

RESOURCEDEF RAM_MODE_S18_S18 {

10  //////////////////////////////////////
   // This is connected to Ports //
   // which has read/write func  //
   //////////////////////////////////////

   FUNCTIONALITY STORAGE;

15  //////////////////////////////////////
   // A Dual Port RAM           //
   //////////////////////////////////////

   TAGS DUALPORT;

   //////////////////////////////////////
   // The number of addressable locations //
   //////////////////////////////////////

25  ATTRIBUTE MAX_LOCATIONS
   {
       attribute + 1024;
   }

30  //////////////////////////////////////
   // The width of each addressable locations //
   //////////////////////////////////////

   ATTRIBUTE MAX_WIDTH
   {
       attribute + 18;
   }

40  //////////////////////////////////////
   // The code to instantiate the RAM        //
   // in mode S18_S18                        //
   //////////////////////////////////////

45  ATTRIBUTE INSTANTIATION
   {
       //////////////////////////////////////
       // component_name is the specific RAM that //
       // needs to be accessed                      //
       //////////////////////////////////////

50  //////////////////////////////////////
       // These signals have to be declared so that//
       //////////////////////////////////////

```

```

// the compiler has them in the Symbol Table//
// before it parses in the MIF interface    //
// code from the ports.                    //
////////////////////////////////////

```

5

```

attribute +
    "input unsigned fixed[10,0] " + component_name +
    "_ADDRB;\n"+
    "input unsigned fixed[16,0] " + component_name +
10  "_DIA;\n"+
    "input unsigned fixed[2,0] " + component_name +
    "_DIPA;\n"+
    "output unsigned fixed[16,0] " + component_name +
    "_DOA;\n"+
15  "output unsigned fixed[2,0] " + component_name +
    "_DOPA;\n"+
    "input unsigned fixed[1,0] " + component_name +
    "_WEA;\n"+
    "input unsigned fixed[1,0] " + component_name +
20  "_ENA;\n";

```

```

attribute +
    "input unsigned fixed[10,0] " + component_name +
    "_ADDRB;\n"+
25  "input unsigned fixed[16,0] " + component_name +
    "_DIB;\n"+
    "input unsigned fixed[2,0] " + component_name +
    "_DIPB;\n"+
    "output unsigned fixed[16,0] " + component_name +
30  "output unsigned fixed[2,0] " + component_name +
    "_DOB;\n"+
    "output unsigned fixed[2,0] " + component_name +
    "_DOPB;\n"+
    "input unsigned fixed[1,0] " + component_name +
    "_WEB;\n"+
35  "input unsigned fixed[1,0] " + component_name +
    "_ENB;\n";

```

```

attribute +
40  "instantiate RAMB16_S18_S18 : " + component_name +
    "(" +
        "DIA = " + component_name + "_DIA," +
        "DIPA = " + component_name + "_DIPA," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOA = " + component_name + "_DOA," +
45  "DOPA = " + component_name + "_DOPA," +
        "ENA = " + component_name + "_ENA," +
        "WEA = " + component_name + "_WEA," +
        "CLKA = " + clock_name + " , " +
        "SSRA = " + reset_name + " , " +
50  "DIB = " + component_name + "_DIB," +
        "DIPB = " + component_name + "_DIPB," +
        "ADDRB = " + component_name + "_ADDRB," +
        "DOB = " + component_name + "_DOB," +
        "DOPB = " + component_name + "_DOPB," +

```


ATTRIBUTE INSTANTIATION

```

5      {
      ////////////////////////////////////////////////////
      // component_name is the specific RAM that  //
      // needs to be accessed                      //
      ////////////////////////////////////////////////////

10     ////////////////////////////////////////////////////
      // These signals have to be declared so that//
      // the compiler has them in the Symbol Table//
      // before it parses in the MIF interface    //
      // code from the ports.                    //
15     ////////////////////////////////////////////////////

      attribute +
          "input unsigned fixed[9,0] " + component_name +
20     "_ADDRB;\n"+
          "input unsigned fixed[32,0] " + component_name +
          "_DIA;\n"+
          "input unsigned fixed[4,0] " + component_name +
          "_DIPA;\n"+
          "output unsigned fixed[32,0] " + component_name +
25     "_DOA;\n"+
          "output unsigned fixed[4,0] " + component_name +
          "_DOPA;\n"+
          "input unsigned fixed[1,0] " + component_name +
          "_WEA;\n"+
30     "input unsigned fixed[1,0] " + component_name +
          "_ENA;\n";

      attribute +
          "input unsigned fixed[9,0] " + component_name +
35     "_ADDRB;\n"+
          "input unsigned fixed[32,0] " + component_name +
          "_DIB;\n"+
          "input unsigned fixed[4,0] " + component_name +
          "_DIPB;\n"+
40     "output unsigned fixed[32,0] " + component_name +
          "_DOB;\n"+
          "output unsigned fixed[4,0] " + component_name +
          "_DOPB;\n"+
          "input unsigned fixed[1,0] " + component_name +
45     "_WEB;\n"+
          "input unsigned fixed[1,0] " + component_name +
          "_ENB;\n";

      attribute +
50     "instantiate RAMB16_S36_S36 : " + component_name +
          "(" +
          "DIA = " + component_name + "_DIA," +
          "DIPA = " + component_name + "_DIPA," +
          "ADDRB = " + component_name + "_ADDRB," +
          "DOA = " + component_name + "_DOA," +

```

```

        "DOPA = " + component_name + "_DOPA," +
        "ENA = " + component_name + "_ENA," +
        "WEA = " + component_name + "_WEA," +
5         "CLKA = " + clock_name + " , " +
        "SSRA = " + reset_name + " , " +
        "DIB = " + component_name + "_DIB," +
        "DIPB = " + component_name + "_DIPB," +
        "ADDRB = " + component_name + "_ADDRB," +
10        "DOB = " + component_name + "_DOB," +
        "DOPB = " + component_name + "_DOPB," +
        "ENB = " + component_name + "_ENB," +
        "WEB = " + component_name + "_WEB," +
        "CLKB = " + clock_name + " , " +
        "SSRB = " + reset_name +
15        ");";
    }
}

20  //////////////////////////////////////
    // A Virtual Non-Existing      //
    // Resource which connects      //
    // the memory to a memory port //
    //////////////////////////////////////
25  RESOURCEDEF VIRTUAL_LINK {

        //////////////////////////////////////
        // This is a virtual bus      //
        //////////////////////////////////////
30

        FUNCTIONALITY TRANSPORT;

        //////////////////////////////////////
        // Tag which indicates to the //
        // compiler that the link is   //
        // virtual                      //
        //////////////////////////////////////
35

        TAGS VIRTUAL;

        //////////////////////////////////////
        // This resource can always   //
        // perform the transport       //
        // functionality               //
        //////////////////////////////////////
40

        ATTRIBUTE CAN_DO
        {
50            attribute + "true";
        }
}

```

```

////////////////////////////////////
// Actual Block SelectRAM resource. There are 24 of //
// these resources in the XC2V250. But since each of//
5 // these RAMs can be configured into 21 different //
// modes, the RDL treats the SelectRAM operating in //
// each of the modes as a virtual resource which //
// USES this actual resource //
////////////////////////////////////
10

RESOURCEDEF BlockSelectRAM {

    //////////////////////////////////
15 // This is connected to Ports //
// which has read/write func //
////////////////////////////////

    FUNCTIONALITY STORAGE;

20

    //////////////////////////////////
// A Dual Port RAM //
////////////////////////////////

25    TAGS DUALPORT;

    //////////////////////////////////
// This resource can always //
// perform the transport //
30 // functionality //
////////////////////////////////

    ATTRIBUTE CAN_DO
    {
35         attribute + "true";
    }
}

40 //////////////////////////////////
// Functionality and Architecture description for the Xc2V250 //
// Reconfigurable part. The Logic Cells are arranged in a 24 x 16 //
// grid. //
////////////////////////////////////

45 RESOURCEDEF FPGA_ADD
{
    FUNCTIONALITY ADD;
    ATTRIBUTE CAN_DO
50     {
        <ADD>
        {
            attribute + "true";
        }
    }
}

```

```

    }
    ATTRIBUTE PIPE_DELAY
    {
5      <ADD>
        {
            attribute + "0";
        }
    }
10
    ATTRIBUTE COMBINATIONAL
    {
        <ADD>
        {
15            attribute + "true";
        }
    }

    ATTRIBUTE NUM_STATES
20    {
        <ADD>
        {
            attribute + "1";
25        }
    }

    ATTRIBUTE INTERFACE
30    {
        <ADD>
        {
            attribute + "state1 : {" + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
35                attribute + in1->constant_value();
            }
            else
            {
                attribute + in1->name();
40            }

            attribute + " + " ;
            if(in2->is_constant() == true)
            {
45                attribute + in2->constant_value();
            }
            else
            {
                attribute + in2->name();
50            }
            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "};";
        }
    }

```



```

    }
}

5 RESOURCEDEF FPGA_SUB
{
    FUNCTIONALITY SUB;
    ATTRIBUTE CAN_DO
    {
10         <SUB>
            {
                attribute + "true";
            }
    }

15     ATTRIBUTE PIPE_DELAY
    {
        <SUB>
        {
20             attribute + "0";
        }
    }

    ATTRIBUTE COMBINATIONAL
25     {
        <SUB>
        {
            attribute + "true";
        }
30     }

    ATTRIBUTE NUM_STATES
    {
35         <SUB>
        {
            attribute + "1";
        }
    }

40     ATTRIBUTE INTERFACE
    {
        <SUB>
        {
45             attribute + "state1 : {";
            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
50             }
            else
            {
                attribute + in1->name();
            }
        }
    }
}

```

```

        attribute + " - " ;
        if(in2->is_constant() == true)
        {
5           attribute + in2->constant_value();
        }
        else
        {
10          attribute + in2->name();
        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
15    }
}

```

```

20    RESOURCEDEF FPGA_MULT
    {
        FUNCTIONALITY MULT;
        ATTRIBUTE CAN_DO
        {
25            <MULT>
            {
                attribute + "true";
            }
        }

30        ATTRIBUTE PIPE_DELAY
        {
            <MULT>
            {
35                attribute + "0";
            }
        }

        ATTRIBUTE COMBINATIONAL
        {
40            <MULT>
            {
                attribute + "true";
            }
        }

45        ATTRIBUTE NUM_STATES
        {
            <MULT>
50            {
                attribute + "1";
            }
        }
    }

```

ATTRIBUTE INTERFACE

```

{
    <MULT>
    {
5       attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
10          attribute + in1->constant_value();
        }
        else
        {
            attribute + in1->name();
        }

15        attribute + " * " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
20        }
        else
        {
            attribute + in2->name();
        }
25        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
        }
30    }
}

```

RESOURCEDEF FPGA_DIV

```

{
35    FUNCTIONALITY DIV;
    ATTRIBUTE CAN_DO
    {
        <DIV>
        {
40          if(in2->is_constant() == true &&
            in2->is_power_of_two() == true)
            {
                attribute + "true";
            }
45          else
            {
                attribute + "false";
            }
        }
50    }
}

```

ATTRIBUTE PIPE_DELAY

```

{
    <DIV>

```

```

        {
            attribute + "0";
        }
    }
5
ATTRIBUTE COMBINATIONAL
{
    <DIV>
    {
10        attribute + "true";
    }
}

ATTRIBUTE NUM_STATES
15 {
    <DIV>
    {
20        attribute + "1";
    }
}

ATTRIBUTE INTERFACE
25 {
    <DIV>
    {
        attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
30        {
            attribute + in1->constant_value();
        }
        else
        {
35        attribute + in1->name();
        }

        attribute + " / " ;
        if(in2->is_constant() == true)
40        {
            attribute + in2->constant_value();
        }
        else
        {
45        attribute + in2->name();
        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
50        attribute + "}";
    }
}
}

```

RESOURCEDEF FPGA_REM

{

 FUNCTIONALITY REM;

 ATTRIBUTE CAN_DO

5

{

 <REM>

{

 if(in2->is_constant() == true &&

 in2->is_power_of_two() == true &&

10

 in1->bitwidth() == out1->bitwidth())

 {

 attribute + "true";

 }

 else

15

 {

 attribute + "false";

 }

 }

 }

20

 ATTRIBUTE PIPE_DELAY

 {

 <REM>

 {

25

 attribute + "0";

 }

 }

ATTRIBUTE COMBINATIONAL

30

{

 <REM>

 {

 attribute + "true";

 }

35

}

ATTRIBUTE NUM_STATES

{

40

 <REM>

 {

 attribute + "1";

 }

}

45

ATTRIBUTE INTERFACE

{

 <REM>

 {

50

 attribute + "state1 : {";

 attribute + out1->name() + " = ";

 if(in1->is_constant() == true)

 {

 attribute + in1->constant_value();

```

    }
    else
    {
        attribute + in1->name();
5        }

        attribute + " rem " ;
        if(in2->is_constant() == true)
        {
10            attribute + in2->constant_value();
        }
        else
        {
            attribute + in2->name();
15        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
20    }
}

25 RESOURCEDEF FPGA_AND
{
    FUNCTIONALITY AND;
    ATTRIBUTE CAN_DO
    {
30        <AND>
        {
            if(in1->bitwidth() == in2->bitwidth() &&
                out1->bitwidth() == in1->bitwidth())
            {
35                attribute + "true";
            }
            else
            {
                attribute + "false";
40            }
        }
    }

    ATTRIBUTE PIPE_DELAY
45    {
        <AND>
        {
            attribute + "0";
        }
50    }

    ATTRIBUTE COMBINATIONAL
    {
        <AND>

```

```

        {
            attribute + "true";
        }
    }
5
    ATTRIBUTE NUM_STATES
    {
        <AND>
10        {
            attribute + "1";
        }
    }

15    ATTRIBUTE INTERFACE
    {
        <AND>
        {
            attribute + "state1 : {";
20            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
            }
25            else
            {
                attribute + in1->name();
            }

            attribute + " and " ;
30            if(in2->is_constant() == true)
            {
                attribute + in2->constant_value();
            }
35            else
            {
                attribute + in2->name();
            }
            attribute + "; \n";
40            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}\"";
        }
    }

45 }

RESOURCEDEF FPGA_OR
{
    FUNCTIONALITY OR;
50    ATTRIBUTE CAN_DO
    {
        <OR>
        {
            if(in1->bitwidth() == in2->bitwidth() &&

```



```

        attribute + in2->constant_value();
    }
    else
    {
        attribute + in2->name();
    }
    attribute + "; \n";
    // To end the list of states
    attribute + "goto NEXTSTATE ;\n";
    attribute + "}";
}
}

15 RESOURCEDEF FPGA_NOT
{
    FUNCTIONALITY NOT;
    ATTRIBUTE CAN_DO
    {
20         <NOT>
        {
            if(in1->is_integer() == true)
            {
                attribute + "false";
            }
            else
            {
                attribute + "true";
            }
        }
    }

    ATTRIBUTE PIPE_DELAY
    {
35         <NOT>
        {
            attribute + "0";
        }
    }

    ATTRIBUTE COMBINATIONAL
    {
40         <NOT>
        {
            attribute + "true";
        }
    }

    ATTRIBUTE NUM_STATES
50     {
        <NOT>
        {
            attribute + "1";
        }
    }
}

```

```

    }
}

ATTRIBUTE INTERFACE
5      {
        <NOT>
        {
            attribute + "state1 : {";
            attribute + out1->name() + " = ";
10         attribute + " ! " ;
            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
            }
15         else
            {
                attribute + in1->name();
            }
            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "};";
        }
    }
25 }

RESOURCEDEF FPGA_XOR
{
    FUNCTIONALITY XOR;
30     ATTRIBUTE CAN_DO
        {
            <XOR>
            {
                if(in1->bitwidth() == in2->bitwidth() &&
35                 out1->bitwidth() == in1->bitwidth())
                {
                    attribute + "true";
                }
            }
            else
40             {
                attribute + "false";
            }
        }
    }
45 }

ATTRIBUTE PIPE_DELAY
{
    <XOR>
    {
50         attribute + "0";
    }
}

```

ATTRIBUTE COMBINATIONAL

```

    {
        <XOR>
        {
            attribute + "true";
5        }
    }

ATTRIBUTE NUM_STATES
10    {
        <XOR>
        {
            attribute + "1";
15    }
    }

ATTRIBUTE INTERFACE
    {
        <XOR>
20        {
            attribute + "state1 : {";
            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
25                attribute + in1->constant_value();
            }
            else
            {
30                attribute + in1->name();
            }

            attribute + " xor " ;
            if(in2->is_constant() == true)
            {
35                attribute + in2->constant_value();
            }
            else
            {
40                attribute + in2->name();
            }
            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}";
45        }
    }
}

RESOURCEDEF FPGA_EQ
50    {
        FUNCTIONALITY EQ;
        ATTRIBUTE CAN_DO
        {
            <EQ>

```

```

{
    if(in1->is_integer() == true && in2->is_integer() == true
&&
        out1->is_boolean() == true)
5      {
          attribute + "true";
        }
      else
10     {
        true &&
          if(in1->is_fixed() == true && in2->is_fixed() ==
            out1->is_boolean() == true)
15          {
              attribute + "true";
            }
          else
            {
20              if(in1->is_constant() == true ||
                  in2->is_constant() == true &&
                    out1->is_boolean() == true)
                {
                    attribute + "true";
                }
              else
25              {
                  attribute + "false";
                }
            }
        }
    }
30
}

ATTRIBUTE PIPE_DELAY
{
35    <EQ>
    {
        attribute + "0";
    }
}

40
ATTRIBUTE COMBINATIONAL
{
    <EQ>
    {
45        attribute + "true";
    }
}

ATTRIBUTE NUM_STATES
50 {
    <EQ>
    {
        attribute + "1";
    }
}

```

```

    }
}

```

ATTRIBUTE INTERFACE

```

5      {
          <EQ>
          {
              attribute + "state1 : {";
              attribute + out1->name() + " = ";
10         if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
            }
            else
15         {
                attribute + in1->name();
            }

            attribute + " == " ;
20         if(in2->is_constant() == true)
            {
                attribute + in2->constant_value();
            }
            else
25         {
                attribute + in2->name();
            }
            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
30         attribute + "}\"";
        }
    }
35 }

RESOURCEDEF FPGA_NEQ
{
    FUNCTIONALITY NEQ;
    ATTRIBUTE CAN_DO
40     {
        <NEQ>
        {
            if(in1->is_integer() == true && in2->is_integer() ==
45         true &&
                out1->is_boolean() == true)
            {
                attribute + "true";
            }
            else
50         {
                if(in1->is_fixed() == true && in2->is_fixed() ==
                true &&
                    out1->is_boolean() == true)
                {

```

```

        attribute + "true";
    }
    else
    {
        attribute + "false";
    }
}

}

}

ATTRIBUTE PIPE_DELAY
{
    <NEQ>
    {
        attribute + "0";
    }
}

ATTRIBUTE COMBINATIONAL
{
    <NEQ>
    {
        attribute + "true";
    }
}

ATTRIBUTE NUM_STATES
{
    <NEQ>
    {
        attribute + "1";
    }
}

ATTRIBUTE INTERFACE
{
    <NEQ>
    {
        attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
            attribute + in1->constant_value();
        }
        else
        {
            attribute + in1->name();
        }

        attribute + " \\= " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
        }
    }
}

```

```

        else
        {
            attribute + in2->name();
        }
5      attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "};
    }
10  }

RESOURCEDEF FPGA_GT
{
15  FUNCTIONALITY GT;
    ATTRIBUTE CAN_DO
    {
        <GT>
        {
20          if(out1->is_boolean() == true)
          {
              attribute + "true";
          }
          else
25          {
              attribute + "false";
          }
        }
    }
30  }

    ATTRIBUTE PIPE_DELAY
    {
        <GT>
        {
35          attribute + "0";
        }
    }

    ATTRIBUTE COMBINATIONAL
40  {
        <GT>
        {
            attribute + "true";
        }
45  }

    ATTRIBUTE NUM_STATES
    {
        <GT>
50  {
            attribute + "1";
        }
    }

```

```

ATTRIBUTE INTERFACE
{
    <GT>
    {
5        attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
10            attribute + in1->constant_value();
        }
        else
        {
            attribute + in1->name();
        }

15        attribute + " > " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
20        }
        else
        {
            attribute + in2->name();
        }
25        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}" ;
        }
30    }
}

RESOURCEDEF FPGA_GE
{
35    FUNCTIONALITY GE;
    ATTRIBUTE CAN_DO
    {
        <GE>
        {
40            if(out1->is_boolean() == true)
            {
                attribute + "true";
            }
            else
45            {
                attribute + "false";
            }
        }
    }
50
    ATTRIBUTE PIPE_DELAY
    {
        <GE>
        {

```



```

        attribute + "0";
    }
}

5  ATTRIBUTE COMBINATIONAL
    {
        <GE>
        {
            attribute + "true";
10     }
    }

    ATTRIBUTE NUM_STATES
    {
15     <GE>
        {
            attribute + "1";
        }
    }

20     ATTRIBUTE INTERFACE
        {
            <GE>
            {
25                 attribute + "state1 : {";
                    attribute + out1->name() + " = ";
                    if(in1->is_constant() == true)
                        {
30                         attribute + in1->constant_value();
                        }
                    else
                        {
                            attribute + in1->name();
                        }
            }

            attribute + " >= " ;
            if(in2->is_constant() == true)
            {
                attribute + in2->constant_value();
40             }
            else
            {
                attribute + in2->name();
            }

            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}";
            }
50     }
    }

RESOURCEDEF FPGA_LT
{

```

```

FUNCTIONALITY LT;
ATTRIBUTE CAN_DO
{
    <LT>
5      {
        if(out1->is_boolean() == true)
        {
            attribute + "true";
        }
10      else
        {
            attribute + "false";
        }
    }
15  }

ATTRIBUTE PIPE_DELAY
{
    <LT>
20      {
        attribute + "0";
    }
}

25  ATTRIBUTE COMBINATIONAL
    {
        <LT>
        {
            attribute + "true";
30      }
    }

    ATTRIBUTE NUM_STATES
    {
35      <LT>
        {
            attribute + "1";
        }
    }
40  }

    ATTRIBUTE INTERFACE
    {
        <LT>
        {
45      attribute + "state1 : {";
            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
50      }
            else
            {
                attribute + in1->name();
            }
        }
    }

```

```

        attribute + " < " ;
        if(in2->is_constant() == true)
        {
5           attribute + in2->constant_value();
        }
        else
        {
10          attribute + in2->name();
        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
15    }
    }
}

RESOURCEDEF FPGA_LE
20 {
    FUNCTIONALITY LE;
    ATTRIBUTE CAN_DO
    {
        <LE>
25    {
        if(out1->is_boolean() == true)
        {
            attribute + "true";
        }
        else
30    {
            attribute + "false";
        }
        }
    }
35 }

    ATTRIBUTE PIPE_DELAY
    {
        <LE>
40    {
        attribute + "0";
        }
    }

45 ATTRIBUTE COMBINATIONAL
    {
        <LE>
        {
            attribute + "true";
50    }
    }

    ATTRIBUTE NUM_STATES
    {

```

```

        <LE>
        {
            attribute + "1";
        }
5    }

    ATTRIBUTE INTERFACE
    {
        <LE>
10    {
            attribute + "state1 : {";
            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
15                attribute + in1->constant_value();
            }
            else
            {
                attribute + in1->name();
20            }

            attribute + " <= " ;
            if(in2->is_constant() == true)
            {
25                attribute + in2->constant_value();
            }
            else
            {
                attribute + in2->name();
30            }
            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "};\n";
35        }
    }
}

RESOURCEDEF FPGA_SLL
40 {
    FUNCTIONALITY SLL;
    ATTRIBUTE CAN_DO
    {
        <SLL>
45    {
        if(in2->is_integer() == true && in1->is_fixed() == true
&&
        out1->is_fixed() == true &&
        in2->is_power_of_two() == true &&
50    in1->bitwidth() == out1->bitwidth())
        {
            attribute + "true";
        }
        else
    }
}

```

```

        {
            attribute + "false";
        }
    }
5    }

    ATTRIBUTE PIPE_DELAY
    {
        <SLL>
10    {
        attribute + "0";
        }
    }

15    ATTRIBUTE COMBINATIONAL
    {
        <SLL>
        {
            attribute + "true";
20    }
    }

    ATTRIBUTE NUM_STATES
    {
25    <SLL>
        {
            attribute + "1";
        }
    }

30    ATTRIBUTE INTERFACE
    {
        <SLL>
        {
35    attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
            attribute + in1->constant_value();
40    }
        else
        {
            attribute + in1->name();
        }

45    attribute + " sll " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
50    }
        else
        {
            attribute + in2->name();
        }
    }
}

```

```

        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "};";
5      }
    }

10  RESOURCEDEF FPGA_SLR
    {
        FUNCTIONALITY SLR;
        ATTRIBUTE CAN_DO
            {
15          <SLR>
            {
                if(in2->is_integer() == true && in1->is_fixed() == true
&&
                out1->is_fixed() == true &&
20          in2->is_power_of_two() == true &&
                in1->bitwidth() == out1->bitwidth())
                {
                    attribute + "true";
                }
25          else
                {
                    attribute + "false";
                }
            }
        }
30    }

        ATTRIBUTE PIPE_DELAY
        {
            <SLR>
35          {
                attribute + "0";
            }
        }

40    ATTRIBUTE COMBINATIONAL
        {
            <SLR>
            {
                attribute + "true";
45          }
        }

        ATTRIBUTE NUM_STATES
        {
50          <SLR>
            {
                attribute + "1";
            }
        }
    }

```

ATTRIBUTE INTERFACE

```

{
    <SLR>
5      {
        attribute + "state1 : {";
        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
10          attribute + in1->constant_value();
        }
        else
        {
15          attribute + in1->name();
        }

        attribute + " srl " ;
        if(in2->is_constant() == true)
        {
20          attribute + in2->constant_value();
        }
        else
        {
25          attribute + in2->name();
        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
30      }
    }

RESOURCEDEF FPGA_SLA
35  {
    FUNCTIONALITY SLA;
    ATTRIBUTE CAN_DO
    {
        <SLA>
40      {
        if(in2->is_integer() == true && in1->is_fixed() == true
&&
        out1->is_fixed() == true &&
45      in2->is_power_of_two() == true &&
        in1->bitwidth() == out1->bitwidth())
        {
            attribute + "true";
        }
        else
50      {
            attribute + "false";
        }
    }
    }
}

```

```

    ATTRIBUTE PIPE_DELAY
    {
        <SLA>
5        {
            attribute + "0";
        }
    }

10  ATTRIBUTE COMBINATIONAL
    {
        <SLA>
        {
            attribute + "true";
15        }
    }

    ATTRIBUTE NUM_STATES
    {
20        <SLA>
        {
            attribute + "1";
        }
    }

25  ATTRIBUTE INTERFACE
    {
        <SLA>
        {
30            attribute + "state1 : {";
            attribute + out1->name() + " = ";
            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
35            }
            else
            {
                attribute + in1->name();
            }

40            attribute + " sla " ;
            if(in2->is_constant() == true)
            {
                attribute + in2->constant_value();
45            }
            else
            {
                attribute + in2->name();
            }
50            attribute + "; \n";
            // To end the list of states
            attribute + "goto NEXTSTATE ;\n";
            attribute + "}\"";
        }
    }

```



```

    }
}

RESOURCEDEF FPGA_SRA
5 {
    FUNCTIONALITY SRA;
    ATTRIBUTE CAN_DO
    {
        <SRA>
10 {
    if(in2->is_integer() == true && in1->is_fixed() == true
&&
    out1->is_fixed() == true &&
    in2->is_power_of_two() == true &&
15 in1->bitwidth() == out1->bitwidth())
    {
        attribute + "true";
    }
    else
20 {
        attribute + "false";
    }
    }
}

25 ATTRIBUTE PIPE_DELAY
{
    <SRA>
    {
30 attribute + "0";
    }
}

ATTRIBUTE COMBINATIONAL
35 {
    <SRA>
    {
        attribute + "true";
    }
40 }

ATTRIBUTE NUM_STATES
{
    <SRA>
45 {
        attribute + "1";
    }
}

50 ATTRIBUTE INTERFACE
{
    <SRA>
    {
        attribute + "state1 : {";
    }
}

```

```

        attribute + out1->name() + " = ";
        if(in1->is_constant() == true)
        {
            attribute + in1->constant_value();
        }
        else
        {
            attribute + in1->name();
        }

        attribute + " sra " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
        }
        else
        {
            attribute + in2->name();
        }
        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "};
    }
}

RESOURCEDEF FPGA_ROL
{
    FUNCTIONALITY ROL;
    ATTRIBUTE CAN_DO
    {
        <ROL>
        {
            if(in2->is_integer() == true && in1->is_fixed() == true
            &&
                out1->is_fixed() == true &&
                in2->is_power_of_two() == true &&
                in1->bitwidth() == out1->bitwidth())
            {
                attribute + "true";
            }
            else
            {
                attribute + "false";
            }
        }
    }
}

ATTRIBUTE PIPE_DELAY
{
    <ROL>
    {
        attribute + "0";
    }
}

```

```

    }
}

ATTRIBUTE COMBINATIONAL
5  {
    <ROL>
    {
        attribute + "true";
    }
10 }

ATTRIBUTE NUM_STATES
{
    <ROL>
15 {
    attribute + "1";
    }
}

20 ATTRIBUTE INTERFACE
{
    <ROL>
    {
        attribute + "state1 : {";
        attribute + out1->name() + " = ";
25 if(in1->is_constant() == true)
        {
            attribute + in1->constant_value();
        }
        else
30 {
            attribute + in1->name();
        }

        attribute + " rol " ;
        if(in2->is_constant() == true)
35 {
            attribute + in2->constant_value();
        }
        else
40 {
            attribute + in2->name();
        }

        attribute + "; \n";
        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
45 }
    }
}

50 }

RESOURCEDEF FPGA_ROR
{
    FUNCTIONALITY ROR;

```

```

    ATTRIBUTE CAN_DO
    {
        <ROR>
        {
5          if(in2->is_integer() == true && in1->is_fixed() == true
    &&
            out1->is_fixed() == true &&
            in2->is_power_of_two() == true &&
            in1->bitwidth() == out1->bitwidth())
10          {
                attribute + "true";
            }
            else
            {
15                attribute + "false";
            }
        }
    }

20    ATTRIBUTE PIPE_DELAY
    {
        <ROR>
        {
            attribute + "0";
25        }
    }

    ATTRIBUTE COMBINATIONAL
    {
30        <ROR>
        {
            attribute + "true";
        }
    }

35    ATTRIBUTE NUM_STATES
    {
        <ROR>
        {
40            attribute + "1";
        }
    }

    ATTRIBUTE INTERFACE
45    {
        <ROR>
        {
            attribute + "state1 : {";
            attribute + out1->name() + " = ";
50            if(in1->is_constant() == true)
            {
                attribute + in1->constant_value();
            }
            else

```

```

        {
            attribute + in1->name();
        }

5         attribute + " ror " ;
        if(in2->is_constant() == true)
        {
            attribute + in2->constant_value();
        }
10        else
        {
            attribute + in2->name();
        }
        attribute + "; \n";
15        // To end the list of states
        attribute + "goto NEXTSTATE ;\n";
        attribute + "}";
    }
}
20 }

```

```

RESOURCEDEF FPGA
{
    FUNCTIONALITY RECONFIG;
25
    //////////////////////////////////////
    // Total number of CLBs //
    //////////////////////////////////////
    ATTRIBUTE AREA
30    {
        attribute + "4032";
    }
}

35
////////////////////////////////////
// XC2V250 Architecture //
////////////////////////////////////

UNITDEF XC2V250 {
40
    //////////////////////////////////////
    // This is not an actual resource //
    // but rather a placeholder for //
    // component declarations //
45    //////////////////////////////////////

    RESOURCE GLOBAL declarations;
    RESOURCE UNSIGNED_DIVIDER udiv;
    RESOURCE SIGNED_DIVIDER sdiv;
50
    //////////////////////////////////////
    // The embedded multipliers //
    //////////////////////////////////////

```

```

RESOURCE MULT18X18 embedmults[ NUM_MULT_COLUMNS *
                                NUM_EMBED_MULT_PER_COLUMN ];

    //////////////////////////////////////
    // The synchronous embedded      //
5  // multipliers                    //
    //////////////////////////////////////

RESOURCE MULT18X18S syn_embedmults[ NUM_MULT_COLUMNS *
                                    NUM_EMBED_MULT_PER_COLUMN ];

    //////////////////////////////////////
    // The Reconfigurable array      //
    // in the XC2V250                 //
15  //////////////////////////////////////

RESOURCE FPGA fpga_part;

    //////////////////////////////////////
    // List of resources              //
    // which use the FPGA             //
20  //////////////////////////////////////

RESOURCE FPGA_ADD fpga_add;
RESOURCE FPGA_SUB fpga_sub;
25 RESOURCE FPGA_MULT fpga_mult;
RESOURCE FPGA_DIV fpga_div;
RESOURCE FPGA_REM fpga_rem;
RESOURCE FPGA_AND fpga_and;
30 RESOURCE FPGA_OR fpga_or;
RESOURCE FPGA_NOT fpga_not;
RESOURCE FPGA_XOR fpga_xor;
RESOURCE FPGA_EQ fpga_eq;
RESOURCE FPGA_NEQ fpga_neq;
35 RESOURCE FPGA_GT fpga_gt;
RESOURCE FPGA_LT fpga_lt;
RESOURCE FPGA_GE fpga_ge;
RESOURCE FPGA_LE fpga_le;
RESOURCE FPGA_SLL fpga_sll;
40 RESOURCE FPGA_SLR fpga_slr;
RESOURCE FPGA_SLA fpga_sla;
RESOURCE FPGA_SRA fpga_sra;
RESOURCE FPGA_ROL fpga_rol;
45 RESOURCE FPGA_ROR fpga_ror;

    //////////////////////////////////////
    // The 24 Block SelectRAMs in the //
    // XC2V250                         //
50  //////////////////////////////////////

RESOURCE BlockSelectRAM block_select_ram[NUM_RAM];

    //////////////////////////////////////
    // A XC2V250 Block SelectRAM which is configured to work //

```

```

// at MODE S18_S18 with aspect ratio of both ports as //
// A: 16,384 x 1 bit and B: 16,384 X 1 bit //
////////////////////////////////////

5  RESOURCE RAM_MODE_S1_S1 ram_s1_s1[NUM_RAM];
   RESOURCE RAM_PORT_A_S1_S1 porta_s1_s1[NUM_RAM];
   RESOURCE RAM_PORT_B_S1_S1 portb_s1_s1[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinka_s1_s1[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s1_s1[NUM_RAM];

10

////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S18_S18 with aspect ratio of both ports as //
// A: 8192 x 2 bit and B: 8192 X 2 bit //
15  //////////////////////////////////////

   RESOURCE RAM_MODE_S2_S2 ram_s2_s2[NUM_RAM];
   RESOURCE RAM_PORT_A_S2_S2 porta_s2_s2[NUM_RAM];
20  RESOURCE RAM_PORT_B_S2_S2 portb_s2_s2[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinka_s2_s2[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s2_s2[NUM_RAM];

25  //////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S18_S18 with aspect ratio of both ports as //
// A: 4096 x 4 bit and B: 4096 X 4 bit //
30  //////////////////////////////////////

   RESOURCE RAM_MODE_S4_S4 ram_s4_s4[NUM_RAM];
   RESOURCE RAM_PORT_A_S4_S4 porta_s4_s4[NUM_RAM];
   RESOURCE RAM_PORT_B_S4_S4 portb_s4_s4[NUM_RAM];
35  RESOURCE VIRTUAL_LINK vlinka_s4_s4[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s4_s4[NUM_RAM];

40  //////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S18_S18 with aspect ratio of both ports as //
// A: 2048 x 9 bit and B: 2048 X 9 bit //
45  //////////////////////////////////////

   RESOURCE RAM_MODE_S9_S9 ram_s9_s9[NUM_RAM];
   RESOURCE RAM_PORT_A_S9_S9 porta_s9_s9[NUM_RAM];
   RESOURCE RAM_PORT_B_S9_S9 portb_s9_s9[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinka_s9_s9[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s9_s9[NUM_RAM];

50  //////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S18_S18 with aspect ratio of both ports as //
// A: 1024 x 18 bit and B: 1024 X 18 bit //
////////////////////////////////////

```

```

5  RESOURCE RAM_MODE_S18_S18 ram_s18_s18[NUM_RAM];
   RESOURCE RAM_PORT_A_S18_S18 porta_s18_s18[NUM_RAM];
   RESOURCE RAM_PORT_B_S18_S18 portb_s18_s18[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinka_s18_s18[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s18_s18[NUM_RAM];

10  //////////////////////////////////////
   // A XC2V250 Block SelectRAM which is configured to work //
   // at MODE S36_S36 with aspect ratio of both ports as    //
   // A: 512 x 36 bit and B: 512 X 36 bit                    //
   //////////////////////////////////////

15  RESOURCE RAM_MODE_S36_S36 ram_s36_s36[NUM_RAM];
   RESOURCE RAM_PORT_A_S36_S36 porta_s36_s36[NUM_RAM];
   RESOURCE RAM_PORT_B_S36_S36 portb_s36_s36[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinka_s36_s36[NUM_RAM];
   RESOURCE VIRTUAL_LINK vlinkb_s36_s36[NUM_RAM];

20  //////////////////////////////////////
   // The RAM resource at various modes is always connected //
   // to 2 ports. Since this connection is always present    //
   // as the RAM implies its 2 ports, we have created a      //
25  // virtual routing resource for connecting the RAM to    //
   // its ports.                                              //
   //////////////////////////////////////

   for(i = 0;i < NUM_RAM;i = i + 1)
30  {
       RCONNECT(ram_s1_s1[i],porta_s1_s1[i],vlinka_s1_s1[i]);
       RCONNECT(ram_s1_s1[i],portb_s1_s1[i],vlinkb_s1_s1[i]);
   }

35  for(i = 0;i < NUM_RAM;i = i + 1)
   {
       RCONNECT(ram_s2_s2[i],porta_s2_s2[i],vlinka_s2_s2[i]);
       RCONNECT(ram_s2_s2[i],portb_s2_s2[i],vlinkb_s2_s2[i]);
   }

40  for(i = 0;i < NUM_RAM;i = i + 1)
   {
       RCONNECT(ram_s4_s4[i],porta_s4_s4[i],vlinka_s4_s4[i]);
       RCONNECT(ram_s4_s4[i],portb_s4_s4[i],vlinkb_s4_s4[i]);
45  }

   for(i = 0;i < NUM_RAM;i = i + 1)
   {
       RCONNECT(ram_s9_s9[i],porta_s9_s9[i],vlinka_s9_s9[i]);
50  RCONNECT(ram_s9_s9[i],portb_s9_s9[i],vlinkb_s9_s9[i]);
   }

   for(i = 0;i < NUM_RAM;i = i + 1)
   {

```



```

1)      for(i = 0;i< NUM_MULT_COLUMNS * NUM_EMBED_MULT_PER_COLUMN;i = i +
        {
5         USES(syn_embedmults[i],embedmults[i]);
        }
    }

```